NIST GCR 99-781

XML Representation Methods for EXPRESS-Driven Data

United States Department of Commerce
Technology Administration
National Institute of Standards and Technology

NIST GCR 99-781

XML Representation Methods for EXPRESS-Driven Data

Prepared for
U.S. Department of Commerce
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001

By
W. Eliot Kimber
ISOGEN International Corporation

Purchase Order 43NANB911557

November 1999

XML Representation Methods for EXPRESS-Driven Data

Prepared For:	National Institute of Standards and Technology

Prepared By:	W. Eliot Kimber ISOGEN International Corp. eliot@isosgen.com www.isogen.com

Created On: 27 Sept 1999

Last Revised: 1 Nov 1999

Approvals:

• Customer Contact: Joshua Lubell _____ Date: _____

• Director of Services: Renee Swank _____ Date: _____

This document provides an analysis of three methods for the representation of EXPRESS-driven data using XML syntax: the OMG's XMI proposal, the PDML early bound approach, and the late binding approach being developed as Part 28 of ISO 10303.

Table of Contents

About This Document

This document is a technical report covering the complementary but distinct technologies of EXPRESS and XML. As such, it involves the use of a great deal of technical jargon. In this document, technical terms are identified on first use like so: **jargon term** and are defined informally in the Glossary of Technical Terms. Please refer to the appropriate standards for the official definitions of these terms.

The EXPRESS and XML standards present some unfortunate terminology clashes. In particular, the term "entity" has completely different meanings in EXPRESS and XML. In this document, the XML term "entity" is replaced by the term "abstract storage object", which is what an XML entity is.

1. Executive Overview

The STEP standard, ISO 10303, has become the predominant international standard for the definition, management, and interchange of "product" data, being used in a wide variety of industries, from aerospace to shipbuilding to oil and gas to power generation. While it began with the specific goal of enabling the interchange of 3-D CAD drawings, ISO 10303 has evolved into a rich set of general technologies for the description and management of complex data objects. Central to the standard is the EXPRESS data modeling language. The EXPRESS language is a completely generic facility for the definition of entity-attribute (or entity-relationship) models.

NOTE: While the term "EXPRESS" simply refers to the modeling language defined in ISO 10303 Part 11, this report uses "EXPRESS" as a shorthand for ISO 10303 when talking about the various 10303 facilities of relevance to this discussion. Thus, a term such as "EXPRESS repository" should be read as "a repository conforming to the relevant facilities of ISO 10303 and governed by an EXPRESS model". This usage within this report reflects common informal usage within the ISO 10303 community itself.

EXPRESS is used to define data models or **schemas** through the definition of **entity types** and the allowed relationships among them. It uses a fairly classic entity-attribute paradigm in which entities are simply collections of **attributes**. EXPRESS schemas are combined to form **models**. Once defined, a model can be instantiated to create **populations** of **entity instances**. A computer system that manages entity instances is referred to generically as a **repository** or an **EXPRESS repository**.

As defined in the standard, the entities within a repository are abstract data objects with no defined physical representation. This leaves implementors free to use any technology or approach they wish when implementing repository software. However, it leaves open the problem of interchanging data objects between repositories or between EXPRESS repositories and repositories built around other technologies. ISO 10303 solves this problem through a character-based serialization syntax defined in ISO 10303 Part 21 and generically referred to as "Part 21". The Part 21 syntax predates XML by a number of years. It provides a fairly simple purpose-built syntax that is sufficient for the task of representing entities as a character stream. This syntax is not extensible in any defined way and does not reflect current practice and techniques for defining character syntaxes, such as the use of formal languages, syntax style, naming style, etc.

Given the existence of XML as a published World Wide Web Consortium (W3C) recommendation that has been widely adopted and implemented, and given the age and limitations of Part 21, it is only natural to consider the use of XML for the character-based interchange of EXPRESS entity instances (and possibly, EXPRESS schemas).

The primary requirement satisfied by Part 21 is the ability to write out populations of EXPRESS entities in a character-based representation that can be easily transmitted

using normal networking and communications protocols, such as FTP, e-mail, and HTTP. It must represent entities without loss of information.

XML would appear to be well suited to the task of satisfying these requirements:

- XML defines a generic, robust, and time-tested character syntax for representing structured data objects.
- XML provides facilities for the syntactic validation of documents against formal rules, potentially enabling the validation of some or all of the constraints in EXPRESS schemas using generic XML validation software
- Because XML is inherently extensible and flexible, it should enable new facilities in the interchange representation that Part 21 cannot provide
- Because XML is or will be supported in Web browsers, it should be possible to use Web browser technology to easily view EXPRESS entities in some way (EXPRESS entities can always be viewed by writing software that translates entity instances in some repository into HTML (or some other browsable form such as VRML)
- Because XML is normatively tied to an existing ISO standard, ISO 8879 (SGML), it is an acceptable candidate for full use within other ISO standards without the need for further standardization effort
- Because XML is a hot new technology that is attracting lots of attention, its use in ISO 10303 may help to make STEP and EXPRESS more attractive to new audiences and potential users

The various attempts to apply XML to this problem have demonstrated that all the above are in fact compelling reasons to use XML for EXPRESS data representation. However, these attempts have also demonstrated that it is not as easy to do as it might at first appear, in part because EXPRESS has some features that complicate the problem and because the problem itself is more involved that it might at first appear.

One potential problem with XML is that it is easy to over sell. While XML is a useful technology, it is, ultimately, simply a serialization syntax. It does nothing to solve the problems of data interpretation and processing. In particular, just putting data into XML form does not make it any more understandable or interchangeable than it was before, because the recipient of the data must still have an understanding of what the data is semantically. Providing a more transparent and flexible syntax for the data does not change this fact.

Because XML is completely generic and very flexible in how it can be used, there is no one right way to apply XML to this type of problem. Different approaches will be optimized for different purposes and produce very different designs. For this reason it can be difficult, if not impossible, to determine which is the "best" approach. All of the XML representation schemes evaluated in this report demonstrably meet the requirement of fully representing EXPRESS entity instance populations for the purpose of character-

based interchange. What distinguishes them is largely their starting requirements and the details of the design decisions that informed their development.

The three schemes evaluated are:

- Part 28 Late Binding. This is the official work of the XML Representation of EXPRESS-Driven Data new work item, approved in the spring of 1999 by ISO TC184/SC4. It is being developed by a small team of XML and EXPRESS experts. The author is deputy leader of the Part 28 committee.

- Product Data Markup Language (PDML). This is a project performed by Product Data Integration Technologies, Inc. (PDIT), a systems integrator with many years of experience with the STEP standard. The PDML project is primarily funded by the U.S. Air Force. The markup language part of PDML is an attempt to define a schema-neutral early-bound representation of EXPRESS data, with a focus on capturing as many of the schema constraints in the XML document type rules as possible. The author has also been involved with the development of the early binding approach through two workshops funded by the U.S. Department of Defense (DoD)Joint Electronic Commerce Project Office (JECPO) office.

- XML Metadata Interchange (XMI), a proposal to the Object Management Group in response to an RFP for a "stream-based model interchange format" (SMIF). XMI's focus is on the interchange of models and instances mostly in terms of OMG standards (MOF, UML, CORBA, etc.). The author has not been involved in the development of XMI in any way.

On their technical merits, all are equivalent in that they satisfy the primary requirements as stated and do so without obvious design flaws. The Part 28 late-bound approach and the PDML early-bound approach can be viewed as (and will likely become), different flavors of the same basic approach. Their designs are a direct reflection of ISO 10303 and do not attempt to satisfy requirements outside that scope. In particular, they do not have the requirement or goal to be a completely generic object or model markup scheme. By contrast, XMI is a direct reflection of the OMG standards, in particular, the meta object facility (MOF). However, the MOF is so abstract and general that it is possible to map almost any reasonable data modeling mechanism to it, including EXPRESS. Thus, XMI can be used to represent both EXPRESS schemas and population instances by application of the abstraction mapping facilities of the MOF (however, there may be semantic aspects of EXPRESS that cannot be represented in XMI using a completely automatic mapping).

The chief advantage of the Part 28 and PDML approaches is that they are direct reflections of the STEP standard and are therefore easily understood by anyone familiar with the STEP standard. By contrast, the XMI approach requires several layers of mapping and abstraction between the original EXPRESS model and instance representation and its eventual expression as an XMI document. On the other hand, the XMI mechanism exists and is implemented. Thus, the cost of using it is low, although the cost of understanding the result may be higher.

There is also a very real sense in which there is no need to choose between these approaches. Because there can be a formal and well-defined mapping between EXPRESS and the corresponding OMG formalisms, it must be possible to define automatic processes that will convert documents in one form to either of the other two forms without loss of data content (although there may be loss of semantic expression or ability to validate constraints using generic tools). This suggests that the ISO 10303 community can eat its cake and have it. Part 28 will define an optimized interchange representation form for EXPRESS data that is well suited to use by the STEP community while the existence of XMI and defined mappings from EXPRESS to UML (and thus to the MOF) will enable the easy interchange of EXPRESS-based outside of 10303-specific contexts.

2. General EXPRESS Representation Technical Issues

The EXPRESS language provides a rich and robust language for defining entity/attribute data models for the purpose of then creating instances of the entities in the service of information and data management tasks, such as representing information about the physical components of aircraft, ships, and buildings. Like the STEP standard of which it is a part, the EXPRESS language serves the needs of large-scale industry. To do so it must satisfy a large number of sophisticated and complex requirements. The EXPRESS language is correspondingly sophisticated and complex. In designing the language it was not possible to impose design constraints in order to make the language simple for the sake of convenience or economy of implementation. The EXPRESS language reflects the real complexity of the requirements of constituency it serves.

This complexity imposes some challenges on the task of providing character-based interchange representations of EXPRESS-based data. Some of these challenges are neither obvious nor trivial. The story of efforts to define useful and complete interchange representations for EXPRESS-based data has largely been one of understanding and addressing the key difficult technical challenges posed by the EXPRESS language. Additional challenges come from aspects of the EXPRESS language and related parts of ISO 10303 that are missing, incomplete, under specified, or inconsistently implemented. These include the lack of a formal data model for EXPRESS schemas and data instances, the lack of an addressing model for EXPRESS data instances, no facility for exposing and persisting entity instance identifiers, and different interpretations and opinions on the use and interchange of constants.

The key challenges are:

- Representing multiple-inheritance supertype hierarchies
- Representing entities within a multi-schema model
- Providing a coherent and consistent instantiation model
- Representing and preserving entity identity
- Representing and interchanging schema-defined constants.

All of the challenges stem from the representation of entity instances. The representation of schemas themselves poses no difficult challenges. In fact, because the EXPRESS language is itself a well-defined character syntax, there is no absolute requirement for XML representation of schemas at all. However, it is convenient to have an XML form for schemas and its definition imposes no great development cost.

This section discusses the EXPRESS and XML technologies and provides an overview of the technical challenges that arise from their combination for the purpose of serializing EXPRESS entity instances.

2.1. What EXPRESS Is and Does

EXPRESS is a formal language for the definition of entity-attribute data models. It was originally designed for the definition of standard data models describing 3-D graphical representations of physical objects, i.e., CAD drawings. However, EXPRESS is a completely generic modeling language and can therefore be used to model data objects of any type. The EXPRESS language has two main forms: a character-based syntax and a graphical notation (EXPRESS-G). The graphical notation is a subset of the full EXPRESS language. The EXPRESS language is distinguished from similar modeling languages by a complete and robust constraint language by which complex, algorithmic constraints can be imposed on attribute values and entity populations. The EXPRESS language is completely declarative and implementation independent, making it well suited for the definition of standardized data models. EXPRESS is a data modeling language, not an object modeling language, meaning that it only defines entities and their properties. It does not define methods that might be applied to those entities in an application context.

EXPRESS specifications are organized into **schemas**. An EXPRESS schema is a name space of named data types. Data types may be simple types such as strings and integers or entity types, representing more complex collections of attributes (properties). Schemas can be related together to form **models**.

The EXPRESS language is defined in Part 11 of ISO 10303. The current version of EXPRESS is 1.0. Version 1.5 is currently under development for completion in 1999. Version 1.5 is essentially, a bug fix release, addressing a few key deficiencies and unmet requirements. Also under development is EXPRESS Version 2, which is a more complete overhaul of the language, adding a number of new facilities.

A simple EXPRESS schema looks like this:

```
SCHEMA people_and_pets;

  TYPE name : STRING;
  END_TYPE;

  TYPE pet_animals = SELECT (dog, cat);
  END_TYPE;

  ENTITY animal SUPERTYPE OF (human, dog, cat);
     names : LIST [1:?] OF name;
  END_ENTITY;

  ENTITY human SUBTYPE OF (animal);
     works_for : human;
     pets : OPTIONAL LIST [1:?] OF pet_animals;
  END_ENTITY;
```

```
  ENTITY dog SUBTYPE OF (animal);
     catches_frisbees : BINARY;
     WHERE
        wr1: SIZEOF(USEDIN(a_pet, 'PEOPLE_AND_PETS.HUMAN.PETS')) > 0;
  END_ENTITY;

  ENTITY cat SUBTYPE OF (animal);
     addicted_to_catnip : BINARY;
  END_ENTITY;


END_SCHEMA;
```

The EXPRESS-G form of this schema is:

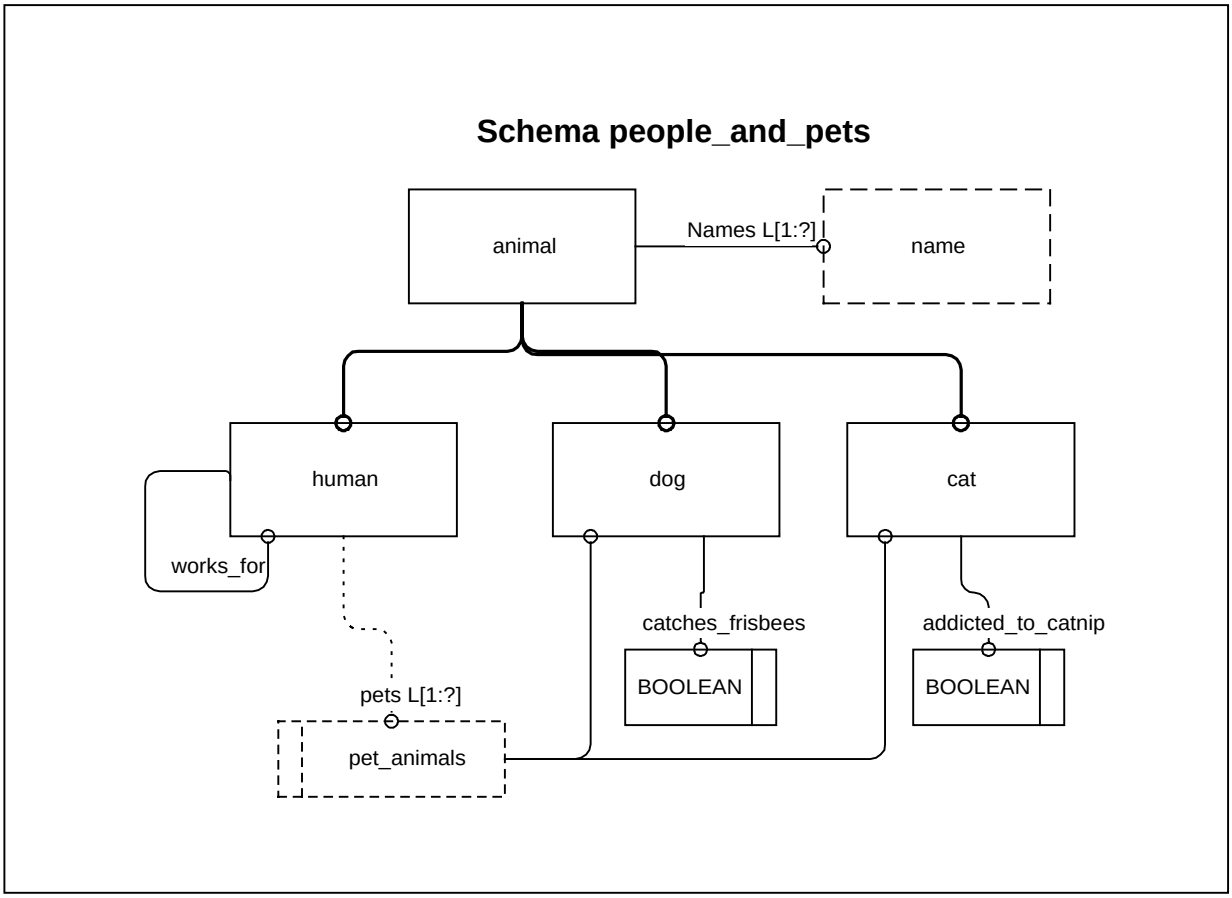

Figure 1 — EXPRESS-G Version of People and Pets Schema

This schema says that people, cats, and dogs are animals. All animals must have one or more names, which are strings of characters. Humans work for other humans and may have cats or dogs as pets. Dogs may be able to catch frisbees and cats may be addicted to catnip.

The EXPRESS language only defines the model, it does not define the mechanism by which instances of this model are represented. The STEP standard provides one representation form, Part 21, and one access API, the STEP Data Access Interface

(SDAI). A Part 21 instance of a population conforming to the people_and_pets schema might look like this:

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION((’People and pets sample instance’)
’1’);
FILE_NAME(’people-and-pets-1.p21’,
’1999-10-15T15:30:00’);
FILE_SCHEMA((’PEOPLE_AND_PETS’));
ENDSEC;
DATA;
#1=human((’W. Eliot Kimber’),#2,(#3,#4,#5))
#2=human((’Renee Swank’),#6,)
#3=cat((’Sigfried’),.T.)
#4=cat((’Bete Noir’, ’Beteski’),.T.)
#5=dog((’Forrest’, ’Dogboy’, ’Noseboy’),.F.)
#6=human((’Carla Corkern’, ’Task Mistress’),#7,(#8, #9))
#7=human((’Lucie Feldstat’),#7,)
#8=dog((’Chauncy’, ’The Chaunce Man’),.F.)
#9=dog((’Chelsie’, ’The Chelster’),.F.)
ENDSEC;
END-ISO-10303-21;
```

In this file, the lines up to the "DATA;" line are header information. Following the data, each line defines a single entity instance. The "#n" numbers serve to uniquely identify each entity instance. Each entity specifies its type. However, the attributes are specified positionally according to the order the attributes are declared in the governing EXPRESS schema.

2.2. What XML Is and Does

XML is a standard for the character representation of hierarchically structured data. The XML specification is a recommendation of the World Wide Web Consortium (W3C), originally published in November 1997. XML is a meta-language or meta-syntax for the creation of domain-specific grammars for the character representation of hierarchical data components.

XML data is organized into **documents**. A document is a set of **elements**, arranged into a singly-rooted tree and bound to a set of syntactic rules, the **document type declaration**. An element may have **attributes** or **content** or both. The content of an element can consist of any combination of data characters and sub-elements, according to the rules defined by the document type declaration. Each element has a defined type, unique within the document. XML elements are roughly analogous to EXPRESS entities but have significantly different characteristics, such that the analogy does not actually permit a direct translation of EXPRESS entity structures to elements and attributes.

XML is a proper subset of SGML, ISO 8879, plus the addition of constraints and conventions that cannot be defined using SGML-provided facilities. XML is nothing more than SGML stripped of its optional features and further constrained such that most of the difficulties in parsing presented by SGML are eliminated. One key distinguishing feature of XML is the ability to omit the document type declaration. In full SGML, it is not possible

in the general case to correctly parse a document without full knowledge of the syntactic constraints defined by the document type declaration (similar to the way that a Part 21 file cannot be decoded without knowledge of the governing schema). XML imposes a set of constraints that make it possible to accurately parse the element tree without knowledge of the document-specific syntax constraints. This is particularly useful with small documents where the volume of data may be significantly smaller than the volume of declarations that describe the rules for the data.

Like SGML, XML is expressly designed for the interchange and archiving of data in a character-based form. An XML document is literally a serialization of the abstract elements that it represents. (Unfortunately, there is not yet a formal definition of what this abstraction is within the set of recommendations provided by the W3C, although there is one for SGML as published in the HyTime standard, ISO/IEC 10744:1997.)

As a generic interchange format, XML has several important characteristics and facilities. First, it is designed to fully support all human languages through the required use of ISO 10646/Unicode character sets. Second, XML provides a storage object abstraction that makes it possible to describe sets of storage objects independent of the details of how those objects are stored at any given time. Third, XML documents are transparent in the sense that their structure is easily human readable, discernible, and modifiable using only the simplest of text editing tools--no specialized software is required. This helps ensure interchange through time. Of course, just being able to distinguish markup from data does not ensure the ability to understand the meaning of the data.

XML is purely a syntax standard. It provides no facilities for defining the semantics of elements. While document type declarations can impose some syntactic constraints, these are relatively limited, certainly in comparison to the full constraint language of EXPRESS. For example, cardinality can in practice only be limited to one, zero or more, or one or more. There is no mechanism for imposing population constraints beyond those definable through simple containment. For example, a document might require that it contain at least one human and at least one animal, but it cannot, by itself, express a constraint that requires one human for every animal.

Thus, while XML has the quality that XML documents are easy to parse (both because the XML specification is clear and as simple as it can be and because there is a wide variety of parsers available, both free and commercial), it cannot be a substitute for data modeling and constraint languages like EXPRESS. XML data, by itself, is "dumb"--it is pure data without methods or inherent processing. XML data does nothing by itself--it waits for something to process it, reconstitute it back into elements and attributes, and apply processing to it.

XML is designed around the fundamental principle of separation of content and presentation. In normal XML practice, documents are as generic as possible, not being tied to any particular style of presentation or processing. Presentation style and behavior is added through separate specifications, generically referred to as **style sheets**. This is essentially a late binding between data entities and methods. While the XML specification does not require this late binding, the ability to have a late binding is one of

the compelling reasons to use XML. This is analogous to EXPRESS, which defines entity types and their relationships, but does not bind methods to entities within the schema. In this sense EXPRESS and XML have the same level of generality, declarativeness, and binding time options.

While SGML was originally designed for the interchange of traditional printed documents and XML was motivated originally by the desire to provide better facilities for delivering traditional documentation in a Web environment, XML is completely generic and can be applied to data of any type. Because XML optimizes the representation of hierarchies, it is usually most naturally applied to data that is inherently hierarchical. However, XML can easily be used to represent data entities that have no natural or primary hierarchy, either by simply creating a flat list of elements or by arbitrarily choosing one of several possible hierarchies. XML does not provide any notion of class or type hierarchies and therefore has no built-in mechanism for reflecting the class hierarchies in EXPRESS models.

Semantically, an XML document is primarily a set of elements. Syntactically, each element is identified by either a pair of start- and end-**tags** or by a simple **empty element tag**, which represents an element that has no syntactic content. Within the start and end tag pair, there may be nested sub-elements or data content, as allowed by the document's DOCTYPE declaration. A complete XML document consists of three parts, two of which are required. The first part of the document is the **XML declaration**, which identifies the data string as being an XML document:

```
<?xml version="1.0"?>
```

Following the XML declaration there may be a DOCTYPE declaration, which contains the declaration of the syntactic rules for the document. The DOCTYPE declaration may be omitted. A DOCTYPE declaration looks like this:

```
<!DOCTYPE mydoc [
  <!-- This is a comment. Declarations for element types and their
attributes go between
          the square brackets: -->
  <!ELEMENT mydoc (a, b) >
  <!ATTLIST mydoc
      id
        ID
        #IMPLIED
  >
]>
```

In this example, the "document type" is "mydoc", which is the name of the required root element. Within the square brackets are the declarations of the element types and their attributes. In this example, the element type "mydoc" is declared to allow two subelements, "a" and "b". It has one attribute named "id", of type "ID" (unique identifier) and it is optional ("#IMPLIED" means that if not specified, the value of the attribute is "implied by the processor"). Note that these are purely syntactic rules. There is no facility in DOCTYPE declarations to express constraints such as the WHERE rule in the schema, except through the use of containment, which is limited, as demonstrated in the next example.

After the DOCTYPE declaration (if present), is the **document instance**, consisting of the root element and any descendant elements or data characters it may contain. There is always exactly one root element.

There is no "typical" XML document. An overly simple and naive XML representation of the EXPRESS entity instances shown above might look like this:

```
<?xml version="1.0"?>
<!DOCTYPE people_and_pets [
  <!ELEMENT people_and_pets
    (human |
     dog |
     cat)*
  >
  <!ELEMENT human
    EMPTY
  >
  <!ATTLIST human
     id
       ID
       #REQUIRED
     names
       CDATA
       #IMPLIED
     works_for
       IDREF
       #REQUIRED
     pets
       IDREFS
       #IMPLIED
  >
  <!ELEMENT dog
    EMPTY
  >
  <!ATTLIST dog
     id
       ID
       #REQUIRED
     names
       CDATA
       #IMPLIED
     catches_frisbees
       (yes | no)
       #REQUIRED
  >
  <!ELEMENT cat
    EMPTY
  >
  <!ATTLIST cat
     id
       ID
       #REQUIRED
     names
       CDATA
       #IMPLIED
     addicted_to_catnip
       (yes | no)
       #REQUIRED
  >
]>
<people_and_pets>
<human id="h1"
```

```
       names="'W. Eliot Kimber'"
       works_for="h2"
       pets="c1 c2 d1"/>
<human id="h2"
       names="'Renee Swank'"
       works_for="h3"/>
<cat id="c1"
     names="'Sigfried'"
     addicted_to_catnip="yes"/>
<cat id="c2"
     names="'Bete Noir' 'Beteski'"
     addicted_to_catnip="yes"/>
<dog id="d1"
     names="'Forrest' 'Dogboy' 'Noseboy'"
     catches_frisbees="no"/>
<human id="h3"
       names="'Carla Corkern' 'Task Mistress'"
       works_for="h4"
       pets="d2 d3"/>
<human id="h4"
       names="'Lucie Feldstat'"
       works_for="h4"/>
<dog id="d2"
     names="'Chauncy' 'The Chaunce Man'
     chases_frisbees="no"/>
<dog id="d3"
     names="'Chelsie' 'The Chelster'"
     chases_frisbees="no"/>
</people_and_pets>
```

This document demonstrates the basic syntax of XML, although not all the features of XML. In this example, each EXPRESS entity has been mapped to a single empty (no content) XML element with all of the EXPRESS attributes mapped to element attributes. Notice, for example, that the supertype "animal" is not directly reflected anywhere in the document, although its "names" attribute is provided by each of the subclasses of animal. References from entity to entity are made with XML ID references, one of XML's two built-in referential mechanisms (the other being references to external abstract storage objects, unfortunately called "entities" in XML). Clearly, there are many aspects of even this simple EXPRESS schema that the above approach cannot capture directly. Note however, that is is comparable to the Part 21 data set. While the XML form is a bit more verbose than the Part 21 file, it is also be bit more transparent, such that the mapping between the EXPRESS schema and the XML representation of the entities is clearer in this XML example than in the Part 21 file, in large part because the attributes are explicitly labeled in the XML, while they are not in the Part 21 data set (in Part 21, instance attributes are mapped to the schema positionally by the order of occurrence of the attributes in the schema).

The entity instances could also be serialized like so:

```
<?xml version="1.0"?>
<!DOCTYPE people_and_pets [
  <!ELEMENT people_and_pets
    (human*)
  >
  <!ELEMENT human
    (name+,
     (cat |
```

```
      dog)*)
  >
  <!ATTLIST human
     id
       ID
       #REQUIRED
     works_for
       IDREF
       #REQUIRED
  >
  <!ELEMENT dog
    (name+)
  >
  <!ATTLIST dog
     id
       ID
       #REQUIRED
     catches_frisbees
       (yes | no)
       #REQUIRED
  >
  <!ELEMENT cat
    (name+)
  >
  <!ATTLIST cat
     id
       ID
       #REQUIRED
     addicted_to_catnip
       (yes | no)
       #REQUIRED
  >
  <!ELEMENT name
    (#PCDATA)
  >
]>
<people_and_pets>
<human id="h1"
       works_for="h2">
 <name>W. Eliot Kimber</name>
 <cat id="c1"
     addicted_to_catnip="yes">
     <name>Sigfried</name>
 </cat>
 <cat id="c2"
      addicted_to_catnip="yes">
  <name>Bete Noir</name>
  <name>Beteski</name>
 </cat>
 <dog id="d1"
      catches_frisbees="no"/>
  <name>Forrest</name>
  <name>Dogboy</name>
  <name>Noseboy</name>
 </dog>
</human>
<human id="h2"
       works_for="h3">
 <name>Renee Swank</name>
<human id="h3"
       works_for="h4">
 <name>Carla Corkern</name>
 <name>Task Mistress</name>
 <dog id="d2"
```

```
     chases_frisbees="no">
  <name>Chauncy</name>
  <name>The Chaunce Man</name>
 </dog>
 <dog id="d3"
     chases_frisbees="no"/>
  <name>Chelsie</name>
  <name>The Chelster</name>
 </dog>
</human>
<human id="h4"
       works_for="h4">
 <name>Lucie Feldstat</name>
</human>
</people_and_pets>
```

In this example, the pointers to cat and dog entities have been replaced with direct containment of the dogs and cats inside the elements for the humans that own them. Likewise, the name attribute has been replaced by "name" subelements. This containment approach works for this particular example because it happens that each pet has exactly one owner. However, the works_for attribute must still be represented using a pointer because many people could work for the same person.

This example also highlights an important aspect of using XML to serial data entities: there are an infinite number of reasonable ways to represent the same data as XML. This means that it is not sufficient to simply receive an XML data set--you must also have the original schema and, in the general, non-trivial case, the algorithm by which the objects were serialized into XML form. Without those two pieces of knowledge, it is impossible to reconstitute the XML data back into its native form with complete accuracy. For example, consider this XML document:

```
<schema0001>
<et000 id="h1"
       works_for="h2">
 <at000>W. Eliot Kimber</at000>
 <et002 id="c1"
     addicted_to_et002nip="yes">
     <at000>Sigfried</at000>
 </et002>
 <et002 id="c2"
      addicted_to_et002nip="yes">
  <at000>Bete Noir</at000>
  <at000>Beteski</at000>
 </et002>
 <et001 id="d1"
      et002ches_frisbees="no"/>
  <at000>Forrest</at000>
  <at000>Dogboy</at000>
  <at000>Noseboy</at000>
 </et001>
</et000>
<et000 id="h2"
       works_for="h3">
 <at000>Renee Swank</at000>
<et000 id="h3"
       works_for="h4">
 <at000>Carla Corkern</at000>
 <at000>Task Mistress</at000>
 <et001 id="d2"
```

```
    chases_frisbees="no">
  <at000>Chauncy</at000>
  <at000>The Chaunce Man</at000>
 </et001>
 <et001 id="d3"
    chases_frisbees="no"/>
  <at000>Chelsie</at000>
  <at000>The Chelster</at000>
 </et001>
</et000>
<et000 id="h4"
       works_for="h4">
 <at000>Lucie Feldstat</at000>
</et000>
</schema0001>
```

This document clearly has the same data content as the one before, but if you did not know that, it would be impossible to infer anything about the entities represented. But, given the mapping between entity type and attribute names in the schema and the arbitrary codes used in the document, you would have no difficulty interpreting the data entities.

These examples are simply to demonstrate that even with a trivial schema and data set, there is a wide variety of reasonable markup approaches one can take.

The examples shown above are all **early bound** with respect to the EXPRESS schema, meaning that element type and attribute names in the XML reflect the entity type and attribute names in the schema. In an early-bound approach, each schema or model will result in a unique document type declaration. Because XML elements will be used to represent both EXPRESS entity instances and attribute instances, either processors must have prior knowledge of the schema (so they know which elements are entities and which are attributes) or there must be some built-in indicator that provides that knowledge. For example, the elements could all have an attribute that labels them as being either entities or attributes:

```
<!ELEMENT human
  (name+,
   (cat |
    dog)*)
>
<!ATTLIST human
   id
     ID
     #REQUIRED
   works_for
     IDREF
     #REQUIRED
   EXPRESS_component_type
     CDATA
     #FIXED "entity"
>
<!ELEMENT name
  (#PCDATA)
>
<!ATTLIST name
   EXPRESS_component_type
     CDATA
```

```
      #FIXED "attribute"
  >
```

But even this is too simple, because in this form of the data, there is no explicit mapping between the attribute "pets" and the cat and dog subelements that are the value of the "pets" attribute. While it is obvious to us as humans reading these examples, it would not be clear to a generalized processor.

This suggests that while it is always possible to define any number of XML representations for a given set of EXPRESS entity instances, only some of those representations will be general enough and complete enough to enable generalized processing with schema-independent tools.

A **late bound** document is one in which the element types reflect the generic components of an EXPRESS repository (entities and attributes) and schema-specific information is carried as attributes or subelements of the XML elements. The advantage of late bound markup approaches is that a single document type definition can work for all possible schemas and entity populations, making the creation of generic tools possible and obvious. The disadvantage is that the schema-specific information may be less obvious to casual observers of the documents. A typical late binding might look like this:

```
<?xml version="1.0"?>
<!DOCTYPE entity_population [
  <!ELEMENT entity_population
    (entity*)
  >
  <!ELEMENT entity_instance
    (attribute*)
  >
  <!ATTLIST entity_instance
     id
        ID
        #REQUIRED
     type_name
        CDATA
        #REQUIRED
  >

  <!ELEMENT attribute
    (#PCDATA | entity_ref)*
  >
  <!ATTLIST attribute
     att_name
        CDATA
        #REQUIRED
  >
  <!ELEMENT entity_ref
    EMPTY
  >
  <!ATTLIST entity_ref
     entity
       IDREF
       #REQUIRED
  >
]>
<entity_population>
 <entity id="h1"
         type_name="human">
```

```
  <attribute att_name="names">’W. Eliot Kimber’</attribute>
  <attribute att_name="works_for">
   <entity_ref entity="h2"/>
  </attribute>
  <attribute att_name="pets">
   <entity_ref entity="c1"/>
   <entity_ref entity="c2"/>
   <entity_ref entity="c3"/>
  </attribute>
 </entity>
 <entity id="h2"
         type_name="human">
  <attribute att_name="names">’Renee SwanK’</attribute>
  <attribute att_name="works_for">
   <entity_ref entity="h3"/>
  </attribute>
 </entity>
 <entity id="h3"
         type_name="human">
  <attribute att_name="names">’Carla Corkern’ ’Task Mistress’</attribute>
  <attribute att_name="works_for">
   <entity_ref entity="h4"/>
  </attribute>
  <attribute att_name="pets">
   <entity_ref entity="d2"/>
   <entity_ref entity="d3"/>
  </attribute>
 </entity>
 <entity id="c1"
         type_name="cat"
  <attribute att_name="names">’Siegfried’</attribute>
  <attribute att_name="addicted_to_catnip">True</attribute>
 </entity>
 <entity id="c2"
         type_name="cat"
  <attribute att_name="names">’Bete Noir’ ’Beteski’</attribute>
  <attribute att_name="addicted_to_catnip">True</attribute>
 </entity>
 <entity id="d1"
         type_name="dog"
  <attribute att_name="names">’Forrest’ ’Dogboy’ ’Noseboy’</attribute>
  <attribute att_name="catches-frisbees">False</attribute>
 </entity>
 <entity id="d2"
         type_name="dog"
  <attribute att_name="names">’Chauncy’ ’Chaunceman’</attribute>
  <attribute att_name="catches-frisbees">False</attribute>
 </entity>
 <entity id="d3"
         type_name="dog"
  <attribute att_name="names">’Chelsie’ ’Chelster’</attribute>
  <attribute att_name="catches-frisbees">False</attribute>
 </entity>
</entity_population>
```

This example is the most verbose but it’s also the most transparent and generic. With the late binding, it is possible to write a single processor that can take any late-bound instance and reconstitute it.

2.3. XML Schema

XML Schema is a companion specification for XML currently under development by the W3C. The purpose of XML Schema is to provide a more robust facility for defining the content rules for XML documents, providing an enhancement to or replacement for the existing XML DTD declaration mechanism. The XML Schema design uses XML syntax (that is, a document) to define the rules for other documents. There appear to be three principal motivations behind the XML Schema effort: use of XML syntax instead of DTD syntax, definition of strong data types for attribute values and element content, and true modularity of document type definitions. XML Schema is not intended to be a generic data or object modeling language in the way that EXPRESS or UML are. It is purely for the purpose of describing the rules for XML documents.

While the idea of XML Schema is valuable and the requirements it attempts to satisfy are legitimate and pressing, there are a number of practical difficulties with the specification itself. The first is that the process of developing the specification is inherently difficult (if not impossible) simply because of the complexity of the problem, the open-ended nature of the solution, the number of different communities of interest involved, and the commercial potentials and implications of the result. These forces conspire to create a standardization environment that is difficult at best. To date, the XML Schema effort has been much slower and contentious than originally anticipated. There is not even a clear agreement on what the core requirements are. There are now competing proprietary solutions in the marketplace (most notably Microsoft's XML Data Reduced (XDR), a trimmed-down version of their original XML Data submission to the W3C).

Another problem is the need to coordinate with other XML specifications. For XML Schema to be truly robust, there must be a definition of what the abstract data model of a schema is (if for no other reason than to enable the addressing of components of schemas). However, the XML family of standards does not provide a generic data modeling or abstract data representation facility. Such a facility is needed for other XML-family specifications as well, including XML itself, XLink, and XPointer. However, there does not appear to be a coherent effort to define such a facility.

The complexity of the problem is also daunting. XML DTDs define relatively simple syntax constraints on documents. The DTD facility is very limited in its functionality. This makes it relatively simple. As soon as the problem is opened up to include issues of modularity, data typing, super and subclassing, and so on, the problem gets much more difficult, the number of possible solutions increases, the need to coordinate and reflect different existing communities and methodologies increases, and so on. This simply serves to complicate the specification task to the point where it would try even the most committed and dedicated working group. Add to this potential spoilers who may want the standard to fail in order to further their own commercial or professional aims, and the chance of success is reduced even further.

In any case, even in the unlikely event that a useful XML Schema specification is developed within the W3C, it is a certainty that it will not be directly applicable to the task

of representing EXPRESS-driven schemas or data instances any more than the existing DTD declaration mechanism is, which is to say, not at all except as a convenient way for defining syntax rules that XML parsers can validate. The XML form of EXPRESS-driven data is a serialized representation of the data, not the data itself. The rules that govern the XML markup cannot be the same rules that govern the things the markup represents. That is, even when using an XML Schema approach, the XML schema describes objects that are elements, attributes, and data characters, not the EXPRESS-defined objects the elements, attributes, and data characters are a serialization of. While it may be possible to design the XML representation such that some of the constraints on the original entity population can be imposed on the serialization and therefore validated using XML validation rules, that is not the same as restating the original EXPRESS schema in XML terms. It is more of a syntax trick.

2.4. Serializing Objects With XML

While data objects are "in memory", that is, in the form on which processing software operates on them, they have no necessary form or order. However, when one program wishes to communicate some set of objects to another program the data objects must serialized so that the data can be coherently transmitted over some communication medium.

Serialization can take many forms: programming APIs, special-purpose protocols such as CORBA and DCOM, or character streams such as 10303 Part 21 files and XML documents. Each approach has its advantages and disadvantages. Programming APIs allow for very fast and efficient interchange at the cost of the specialized programming to connect two systems together through their API. Special-purpose protocols trade some efficiency for generality, increasing the potential interoperability of the system and therefore increasing its maintainability at the cost of some additional communication overhead.

Character-stream-based interchange becomes highly general but is the least efficient communication form: the data must be written out as individual characters, then transmitted, then parsed character-by-character and the objects reconstituted. The entire communication process is completely transparent: at any point, the data being transmitted can be easily examined. The only specialized software involved is the writer and reader, and even the writer can be replaced by humans using text editors (at least for simple cases). The approach is highly general. It also works well when the communication connection is not reliable or persistent, such as when using email or when communicating with the future (archiving data for future use).

This section discusses general issues of serialization and their relationship to the use of XML. While the examples used are EXPRESS, the issues presented here apply to the serialization of all objects.

2.4.1. Why Use XML at ALL?

For the specific task of interchanging EXPRESS entity instances using character strings, Part 21 already satisfies the requirements. This begs the question of why use XML at all? The short answer is that Part 21, while meeting the base requirements, does not necessarily meet all of the requirements for interchange, such as extensibility, availability of reading and writing tools, etc.

This also begs the larger question of why use XML at all for object serialization in any context? The short answer is: because it provides a ready-made, robust syntax. Because XML exists as a well-defined and accepted standard, no-one need ever define a purpose-built syntax again. This saves a small, but significant, part of the design of any character-based interchange format. More important, it helps ensure correct understanding and interpretation of the syntax rules of the format by eliminating the dependence on designers to write clear and unambiguous specifications and implementors to read and understand them.

As demonstrated above, XML can be used to represent data of any sort, hierarchical or not--the only challenge is to define the details of the mapping from the native object structure into its XML representation. This design work is non-trivial, but is usually easier and less error prone than designing a syntax from scratch. In short, it is very low risk to use XML to do object serialization.

2.4.2. Early vs. Late Binding

One of the key design issues in serialization is early vs. late binding. An "early bound" serialization is one in which the named components of the serialization format correspond as directly as possible to the named components of the data objects being serialized. For example, given an entity type of "human" in a data model, an early bound API might provide a "human()" function or object and an early-bound XML representation might provide a element type of "human".

The advantage of early binding is that it tends to make the mapping of the representation back to the model more obvious or intuitive to users and observers. It reduces or eliminates the amount of processing work needed to map from the representation back into the things in memory ("reconstitution"). This tends to make programming easier and less error prone.

One disadvantage of early binding is that details of the mapping process that got you from entity type "human" to element type "human" may be lost or obscured. This may lead to ambiguity if the components being mapped to do not correspond directly to the components being mapped from. This is especially true when there is no single possible mapping from the original to the representation form. For example, in mapping to XML, entity attributes could be mapped to element attributes or to subelements of elements or to elements that occur elsewhere and are pointed to by the elements representing entity

instances. All three mappings are reasonable and unless you know what the mapping is, it may be impossible to divine it unambiguously.

For object serialization with XML, this means that there is no obvious or natural mapping approach, even for a given formalism, such as UML or EXPRESS. This means that data serialized into XML is not really very useful unless the receiver of the data has a complete understanding of its mapping from its original form. Thus, while XML data is, in general, transparent because it is in a human-readable syntax, that transparency can only go so far. In many cases (if not most cases), data transferred in XML format will only be marginally less opaque than the same data transferred using a totally opaque protocol such as DCOM or CORBA.

In late-bound serialization, the named components of the serialization format correspond not to the named components of the model, but to the components of the metamodel for the model. Thus, if an EXPRESS schema has an entity type of "human", its late-bound serialization would be through an element of type "entity" with a "type-name" attribute of "human".

One advantage of late bindings is that they tend to be simpler to define and implement because there are fewer types of things to design, document, and code to. For example, there can be a single late bound XML document type that can represent all possible EXPRESS model instances, rather than one document type per model. From a programming standpoint, this means that a single program will be able to read and reconstitute any late-bound instance (although it will almost certainly have no model-specific knowledge). Late bindings tend to be less ambiguous than early bindings because the late binding makes the type of component (entity, attribute, etc.) being represented completely clear.

A disadvantage of late binding is that it usually takes more programming effort to reconstitute the objects because there is more indirection between the details of the serialization and the original things. However, this effort can usually be encapsulated and contained. Late bound formats also tend to be more verbose than their early-bound equivalents.

Another disadvantage of early bindings is that they are often not complete over the set of all possible data instances. For example, in EXPRESS, there is a class of schemas that do not use multiple inheritance. Such schemas allow a much simpler and natural early-bound XML representation form than can be used in the general case where subtypes may have multiple supertypes. This means that there are two classes of early binding: those that are generic to all possible data instances and those that will only work when some set of constraints on the schema or instance population are in effect. For the purpose of standards, the second class is generally not acceptable. For private or domain-specific purposes, the second class may be either acceptable or, in some use cases, a requirement. This means that early bindings must be clearly identified as being either generic or specific to particular sets of constraints.

In XML, it is possible to have an early-bound form that is as informationally complete as any late-bound form, simply by providing attributes that capture the information that would be conveyed by the late-bound element types. That is, if the early-bound representation of the entity type "human" is the element type "human", that element type can also have an XML attribute that indicates that the element type represents an instance of an entity and not an instance of an entity attribute. These attributes define a syntactic mapping or correspondence between an early-bound form and the equivalent late-bound form.

While this attribute-based early-to-late mapping can be simply a matter of private convention, the SGML Architecture mechanism defined in ISO/IEC 10744:1997 provides a standardized form of this type of mapping that can be used to formally define the relationship between two representation forms.

2.4.3. Use of Attributes and Content

One of the key questions when designing an XML representation of any kind of data is how and when to use element attributes and when to use element content. In XML, elements can have attributes, which, syntactically, are simply unstructured string values specified as part of the element’s start-tag. Element attributes are most useful for simple property values, identifiers, or fixed properties of value primarily to processing software. Because XML attributes are simple, unstructured, string values, they are not generally appropriate for representing the values of EXPRESS entity attributes. That is, there cannot be a generic one-to-one mapping of entity attributes to element attributes, simply because of the syntactic limitations of XML attributes.

One important feature of XML attributes is the ability to have "fixed" attributes. Fixed attributes are attributes that have a fixed value declared in the document’s DOCTYPE declaration. Fixed attributes are convenient because they can be specified once in the DOCTYPE declaration but are automatically propagated to all instances of the element type. They are especially useful for conveying mapping information, such as the fact that a given element represents an entity instance or an attribute instance. They may also be used to convey additional metadata about the element, such as data types, cardinality, etc. Note, however, that fixed attributes are simply a syntactic convenience. The same attributes could be explicitly specified on every element instance. The ability to fix the values in the document type declaration simply reduces the amount of markup that must be transmitted.

Because EXPRESS attributes cannot be mapped to XML attributes in the general case, they must be mapped to complete elements. Because the same syntactic mechanism must be used for both entire entities and attributes of entities, it raises the problem of distinguishing elements that represent entities from elements that represent attributes. In a late-bound markup scheme, this is not a problem, because the element types themselves will convey the distinction. However, in early-bound schemes, there may be ambiguity if an entity type and an attribute have the same name. In the XML markup, two

different element type names must be used. One obvious solution is to use attributes to indicate the semantic role a given element plays.

2.4.4. Use of Containment and Hierarchy

Another design challenge for XML representations is when to use hierarchy and containment. An XML document is explicitly a tree of elements, with each element except the root completely and strictly contained by some other element. One way to think of this is that the XML string representation of a set of data objects provides the opportunity to privileged one of the hierarchies in the data through containment. This is often convenient for human observers, especially when using XML-aware browsers that can take advantage of the containment to aid navigation or presentation.

When representing EXPRESS entity instances, there is no natural hierarchy in the general case: entity instances can and usually do form graphs. While some models may enforce a strict hierarchy or have a primary hierarchy (assembly trees, for example), a generic EXPRESS representation mechanism can have no knowledge of such model-specific organizational structures and therefore cannot depend on them. There are other hierarchies in EXPRESS data, in particular, the subtype/supertype hierarchy.

There is a natural and not unreasonable inclination to use XML containment to reflect either the class hierarchy of entities or the structural hierarchy of entities as reflected by attributes whose values are other entities. Unfortunately, neither can be completely represented using XML element hierarchies.

What we informally call an EXPRESS entity instance is actually a **complex entity instance**. A complex entity instance consists of one or more **partial entity instances**, each one exhibiting the attributes defined by its direct type. In the example above, the entity instance for the human named "Carla Corkern" is a complex entity instance consisting of two partial entity instances, one for the supertype animal and one for the subtype human. The fact that the human partial entity instance is a subtype of the animal partial entity instance could be expressed by having an "animal" element that contains the "names" attribute and a "human" element:

```
<animal>
  <names>’Carla Corkern’</names>
  <human>
    <works_for>Lucy Feldstat</works_for>
  </human>
  <!-- Other properties ommitted for clarity -->
</animal>
```

The problem with this approach is that subtypes may have multiple supertypes. In this case, there is no single hierarchy that can express the sub-to-supertype relationships. Thus an approach that uses containment for single-root supertype trees must provide some other mechanism for representing multiply-rooted supertype trees.

Likewise, using containment to reflect the containment of entities in attribute values works only if entities exist in exactly one property value. If an entity is used in more than one attribute value then only one of the attributes can contain the entity and the others must point to it in some way. This requires either arbitrarily assigning the containment (e.g., the first attribute to name an entity gets to contain it) or there must be some way to infer which attributes are really defining the primarily hierarchical structure in the model. Unfortunately, the EXPRESS language provides no facility for identifying attributes as being primary containers. Thus, in the general case, containment cannot be used to reflect entity-to-entity relationships except in an fairly arbitrary and unsatisfying way.

There is also no reason that both the type hierarchy and attribute-to-entity hierarchy cannot be used together. Given that neither hierarchy can be completely represented using element containment, there does not appear to be a compelling argument for not allowing the overloading of the element hierarchy in this way, given that no processing tool can assign any dependable or unique semantic to containment in any case. It is probably useful for general XML representation schemes to allow the use of hierarchy as an option. However, processing tools should never depend on the use of hierarchy to infer properties of the entity instances, as they cannot depend on its being used or being used consistently or meaningfully.

2.4.5. Single- and Multi-Document Packages

ISO 10303 is reasonably consistent in thinking of a repository of EXPRESS entity instances as a single physical collection. This is reflected in Part 21 in that it does not provide a way to distribute a single population of entities across multiple physical files for interchange.

XML, through it’s more complete and flexible abstract storage object facilities (unfortunately called "entities" as well), makes it possible, if not easy, to distribute a single entity population across multiple documents to form a single multi-file package. This can have a number of advantages for interchange, especially in environments where transmitting very large files is problematic.

Thus, one distinguishing characteristic of XML representation methods is their built-in potential for using multiple documents for a single population.

2.4.6. Object Identification

ISO 10303 stipulates that all entity instances have identity within their repository, meaning that they have some sort of inherent object identifier or name. However, the standard does not say what form this identifier might take, leaving it up to implementations to decide how to manage identity and identification. This means, in part, that there is no defined mechanism for assigning persistent or "public" identifiers to entity instances. This is a problem for interchange because it is often useful, if not necessary, to be able to correlate entity instances in the original repository to the copies of them written

to the interchange file (and, by extension, to the copies created in the target repository, if there is one).

Within an XML document, there is no identity problem because every element has clear and unambiguous identity because it exists at a precise position within the document hierarchy. Every element can be easily referred to simply by specifying its position within the element tree (given a set of rules for forming the tree itself). For convenience, elements can be given XML IDs, which must be unique within the document. They could also be given other attributes or content values that serve to identify them within some context. For example, if elements representing attributes are contained by an element that represents the entity instance that exhibits the attributes and they have element type names reflecting the original EXPRESS attribute names, then the element type names will be unique within the context of that element and can be unambiguously addressed in that context.

Thus, from an XML representation standpoint, there is no object identification problem as XML elements have inherent and obvious identity and can be reliably addressed by a variety of means. However, from an interchange perspective, there is the problem of correlating the elements in the interchange document back to their original instances in the repository from which the interchange document was created (assuming that the original entities instances are not destroyed as soon as the interchange file is written).

There are any number of ways to manage this correlation. One would be to add to the XML representation the original object ID. Another would be to provide a separate mapping table from the original object IDs to the elements in the interchange file. In any case, some form of instance-to-instance mapping is a practical requirement for interchange when there needs to be any degree of tracking.

The problem is complicated by the fact that nothing in the 10303 standard provides a defined and implementation-independent way to address entity instances in repositories. While the SDAI API provides methods for referring to entity instances by their repository-specific object IDs, it does not stipulate how those IDs are formed. In addition, there is no defined way to refer to a repository as an object in order to establish the addressing domain in which a given repository-specific object ID is meaningful. Without this, there is no way to provide standard, implementation-independent addressing of entity instances within repositories. This is a serious problem with the STEP standard. Until it is solved, robust, generic, and reliable correlation of entities across repositories is impossible. Nothing that is put into an XML (or Part 21) interchange package can change this fact.

2.5. Challenges Posed by EXPRESS

The EXPRESS language provides some representation challenges over and above the challenges presented by object representation in general.

These challenges include:

- Multiple supertype hierarchies, as discussed above
- Multi-schema models
- The fact that the EXPRESS instantiation model is underspecified, particularly with respect to constants
- The lack of a complete formal data model for EXPRESS schemas
- Select types
- Name case of EXPRESS constructions

2.5.1. Multiple-Supertype Hierarchies

As discussed above, the ability to have multiple supertypes for a single subtype makes it impossible to use strict containment to represent the supertype hierarchy. In an early-bound representation, it also leads to problems with the naming of element types. If the desire in an early-bound representation is for the element representing the complex entity instance to only allow as its content partial entity instances allowed by the EXPRESS schema, then each unique supertype tree must have a unique element type name. Unfortunately, there is no 10303-defined algorithm for forming this name as it is not a requirement in an "in-memory" repository.

2.5.2. Multi-Schema Models

The EXPRESS language provides facilities by which one schema can use entity and data types from another schema (use-from and reference-from). When one schema uses from or references from another schema, it forms a multi-schema model.

The complication caused by multi-schema models is that the using schema may modify the sub and supertype constraints of the used schema. In a late-bound representation this is not a problem because the late bound markup does not directly reflect the schema constraints. However, in early-bound representations where the entity types and their existence constraints are reflected in the XML document type declarations, the details of the XML content models will depend on the interaction of the two schemas. This means that there cannot be exactly one DTD per schema. Rather, there must be one DTD per unique model. Models are defined by which schema is the first or "hub" schema of the model.

Multi-schema models also imply a requirement that all type names must be bound or bindable to their schema context. In an early-bound representation, element type names cannot be simply the entity type name in the general case; they must also include the containing schema name.

Note that EXPRESS already provides an aliasing facility for the USE FROM and REFERENCE FROM declarations that enables the explicit disambiguation of names in

the context of the using schema. However, this facility does not ensure that names of types used indirectly (because they are required by the directly-used types) will not clash with names in the using schema. Therefore, it is not possible to depend on the use of aliasing to provide unique names.

2.5.3. Representation of Constants

The EXPRESS language allows the definition of constants. Constants may be either simple data values used in expressions or complete entity instances. For constant data values, there is no interchange requirement as long as attribute values are always fully resolved. However, because attribute values can refer to constant entity instances, there is a requirement to interchange constant entity instances to ensure referential integrity and completeness in the interchange package. The key requirement is the identification of entity instances created by constant declarations as being constant. (For example, Part 21 provides no way to distinguish constant entity instances from non-constant instances with the same attribute values).

If ISO 10303 provided an addressing model and syntax by which entity instances in the original repository or in the original schema could be pointed to, it would not be necessary to actually interchange the constant entity instances because they could be pointed to where they exist. However, 10303 does not provide any defined facilities for pointing at entity instances from outside a repository. While the SDAI interface provides a model for pointing to entity instances, it does not provide a syntax for representing those pointers in a character stream. In addition, because the identification method for entity instances is undefined by ISO 10303 (including in the SDAI), there can be no general-purpose syntax for referring to entity instances in repositories.

2.5.4. Lack of Complete Formal Data Model for EXPRESS

Like XML, the EXPRESS language is only formally defined as a syntax through a set of BNF-like productions. This syntactic definition, while rigorous and unambiguous, does not define the abstract objects of which the syntax is a representation. This poses a problem because the XML representation of data requires a mapping of XML-syntax objects to the abstractions they represent (entity, attribute, type definition, etc.). For example, an entity type declaration in an EXPRESS schema defines an entity type. The declaration is the syntactic construct but "entity type" is the abstract construct. An entity instance is an instance of a type, yet there is no formal data model for what an entity type is. There are some partial models for EXPRESS (notably, Phil Spiby’s N228 document from 1995), but there is no complete model published as part of ISO 10303 (it is expected that the EXPRESS 2 definition will include such a model, however).

This lack of a formal data model for EXPRESS schemas requires XML representation designers to infer or invent a model with no guarantee that it is either correct with respect to the standard or consistent with other models of EXPRESS developed by other groups. This is particularly problematic for XMI, which depends on the existence of a formal

abstract data model that can be rigorously mapped to the MOF meta model. Without a solid model for EXPRESS, it is impossible to use XMI in a standards-defined way.

2.5.5. Select Types

When declaring select types it is possible to create multiple, different, paths from the supertypes to the base type. If the attribute value is labeled with the data type, then there is no ambiguity about what the type derivation path is. However, if the attribute value is not labeled with the data type, but is simply stated or labeled only with the leaf type, then it is impossible to know which of the allowed types at the top of the select hierarchy was actually intended for the attribute value.

This issue does not occur in any of the XML representation samples shown in this report. However, it is a problem for Part 21 files and would be a problem for an XML representation that is no more descriptive than Part 21 (that is, an XML representation that depends on reader having full knowledge of the governing schema in order to reconstitute the data values).

3. Metamodels for EXPRESS

Any XML serialization document is a serialization of objects that conform to some abstract data or object model. Thus, in order to define a serialization syntax, you must first understand the abstract data model to be represented.

The presents a problem for the XML representation of EXPRESS: there is no normative metamodel for either EXPRESS schemas or EXPRESS entity instances. There are several different models of varying degrees of completeness that have been developed for different purposes, but no single all-encompassing abstract model. This is a problem because there is no way to test a given representation scheme for correctness with respect to ISO 10303 if there is no standardized abstract model. This was made clear during the recent EXPRESS XML Early Binding workshop when the members of the workshop, all experts in EXPRESS and EXPRESS-based modeling, had fundamental disagreements about what the abstract structure of schemas and schemas were. This meant that the workshop participants had to define an abstract model simply so that they could complete their design work. However, there is no guarantee that this particular model is either the best possible model or consistent with the work being done now and in the future in the definition of EXPRESS 2. Thus there is the serious risk that an XML representation defined today will reflect an abstract model that is inconsistent with the eventual normative model of EXPRESS. Note that Part 22, the SDAI, had the same problem and had to take the same solution. The SDAI designers developed a metamodel for EXPRESS that reflected the needs of the SDAI interface. This metamodel is not complete relative to EXPRESS because the SDAI itself does not reflect all of EXPRESS and therefore it would have been unnecessary (and probably inappropriate) to define a larger model.

This lack of a normative abstract model for EXPRESS also poses a problem for the use of XMI. XMI is predicated on the use of MOF-conforming metamodels. Therefore, for any EXPRESS model or instance to be interchanged using XMI markup, there must first be created a MOF (essentially UML) version of the same model. Unfortunately, there is no normative EXPRESS-to-UML mapping. In addition, for this mapping, there is the same validation problem as with the XML syntax definition: it is impossible to test the correctness of the MOF transliteration of a given EXPRESS model without a corresponding MOF model of the EXPRESS abstractions to which the EXPRESS model and instances must conform. Therefore, there needs to be a MOF model of EXPRESS. This MOF model must of course accurately reflect the true abstract model for EXPRESS. But this model does not yet exist.

Thus, while Part 28 can provide a normative mapping from the (undefined) abstractions that EXPRESS defines into XML (just as Part 22 did) simply because it will be part of the standard, there cannot be a normative mapping of EXPRESS into XMI without a normative mapping of EXPRESS into the MOF. Because this mapping must be an abstraction-to-abstraction mapping, there must be a normative abstraction of EXPRESS to act as the source of the mapping.

For the purposes of this analysis, I have created a UML transliteration of the EXPRESS semantic model of EXPRESS 1 developed by Phil Spiby of Eurostep, UK Ltd. and published in *TC184/SC4 N288*, as well as a UML transliteration of the people-and-pets model. I have also used the UML transliteration of the SDAI model of EXPRESS, provided by David Price of PDES, Inc. and IBM. These models are used as illustrative examples and should not be taken as suggestions for a normative model of EXPRESS. Discussion of what, if any, abstract model of EXPRESS should be used for the Part 28 design will be part of the agenda for the NWI effort at the ISO 10303 meeting New Orleans, 8-12 November 1999.

I use the term "transliteration" to make it clear that these UML models are an attempt to reflect the letter and intent of the EXPRESS models as closely as possible. They do not necessarily reflect the models one would get from using UML directly.

3.1. Difficulties in Transliterating EXPRESS Into UML

The EXPRESS and UML languages are similar in that they are both general-purpose modeling languages. They are obviously comparable for the general task of defining data models. However, at the detail level, there are some significant differences in the two languages that make it difficult to translate models between them. Most of these differences reflect the different requirements sets that led to the languages: EXPRESS is a pure data modeling language from the entity-relationship domain, UML is an object modeling language from the object and implementation domain. Other differences stem from differences in design choice and basic expressiveness of the two languages. Neither language is objectively "better" than the other: they are simply different and designed for different purposes by different groups of people. Nevertheless, the differences in the languages does lead to some practical problems in mapping between EXPRESS and UML/MOF.

In creating the UML models used for this analysis, I identified the following issues for EXPRESS-to-UML mapping:

- UML provides two types of class-to-class relationship: aggregation and association, while EXPRESS provides one: attribute value membership. It is not entirely clear which of these two relationship types should be used to represent entity-valued attributes or exactly how each should be used with respect to the labeling of relationships and roles. For these models, I used a combination of association and aggregation. However, in retrospect, in appears that aggregation is not appropriate and that association should be used exclusively. This is because aggregation in UML implies a containment relationship that EXPRESS provides no direct way to represent (even though it may be clear from the semantics of the model when a given attribute represents semantic containment and when it does not). The use of association vs. aggregation affects the form of the generated XMI markup as the XMI generation rules express aggregation through direct containment of class instance elements within their containing class instances. Association is represented by the use of references. Note that the MOF model imposes the constraint that no class may be contained by more than one other

class. Note that in the model provided by David Price, relationships are defined as aggregations. In private communication, David indicated that he is working on defining an alternative version that uses association instead.

- It was not clear what was the best way to represent explicit inverse relationships. I chose to represent them as separate explicit associations in UML, although it may be more appropriate to use UML's ability to label both ends of an association.
- There is no direct analog of select types in UML because UML does not provide a way to have a single association connect a single source class to multiple target classes. The only representation choices appear to be either to translate a select type into a set of associations, one for each member of the select type, or to create a UML class to represent the select type. I chose the latter because it was clearer visually is a closer semantic match to the intent of select types as there is no obvious mechanism in UML to bind a set of otherwise independent associations together to form a single logical attribute. However, it has the disadvantage that the UML class definition no longer exactly matches the entities in the EXPRESS model.
- It was not clear how to represent the one-of/and-or subtype instantiation constraints in the UML.
- There is no obvious way to represent optional attributes that are not entity-valued. For entity-valued attributes, optionality can be represented by specifying a cardinality of "[0..1]".
- There does not appear to be a way to define new data types in UML and there is definitely no way to do it in the MOF or XMI, as they are explicitly limited to the CORBA data types. One consequence of this is that there is not usually a direct reflection of the data types of attributes values.

Some of these issues may stem from the author's inexperience with UML. The scope of this project did not allow for deep study of UML.

As a result of these issues, the UML models presented here must be taken for what they are: experiments in mapping and nothing more.
The basic mapping from EXPRESS to UML used for the models is:

entity type

UML class

attribute

For simple-valued attributes, maps to a class attribute. For entity-valued attributes, maps to an association relationship where the EXPRESS attribute name is used as the label of the target end of the relationship. Optional attributes are given a starting cardinality of "[1..1]"

select type

An abstract UML class that is a superclass of the members of the select type.

defined type

No direct mapping. Defined type names are used as the attribute type. UML allows this. However, there is no defined mapping into XMI for these types.

aggregates of simple types

No direct or obvious mapping.

schema

UML package

use-from

UML import

3.2. UML Model Of the People and Pets Model

Figure 2. *UML Transliteration of the People and Pets Model*, page 32, shows the UML transliteration of the people and pets model.

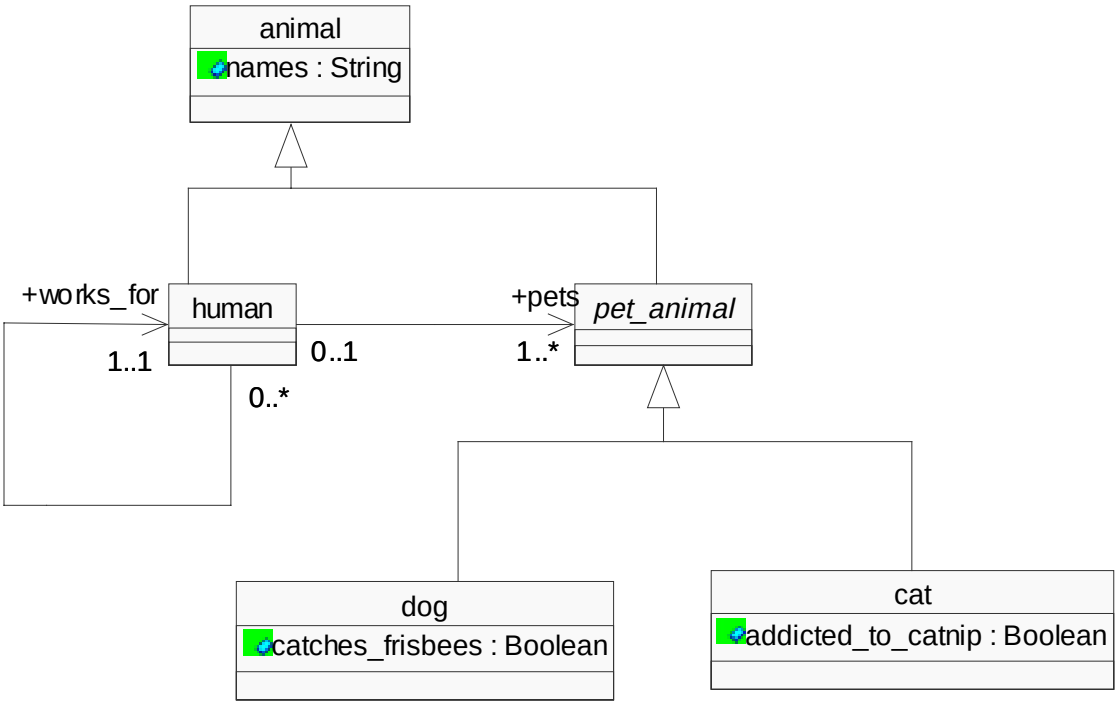

Figure 2 — UML Transliteration of the People and Pets Model

The UML transliteration is fairly straight forward. However, there are some interesting points to note.

First, the select type "pet_animals" becomes an abstract class. This will cause a problem in the XMI generation because the class will be reflected in the markup.

The relationships "works_for" and "pets" become associations in UML. It appears that UML (and XMI) expect associations to have their ends labeled, as the XMI generation rules use the association role names as element type names in representing the associations. However, EXPRESS does not provide for explicit labeling of the roles in the relationships represented by entity-valued attributes. (In EXPRESS, the normal modeling technique for representing labeled relationships is to create an entity type for the relationship.) This suggests that the mapping of EXPRESS attributes to UML attributes and associations is not a clean one. UML does not appear to have a construct that directly reflects EXPRESS' entity-valued attribute concept.

3.3. UML Model from SDAI EXPRESS Meta-Model

Figure 3. *UML Transliteration of SDAI EXPRESS Abstract Model, Part 1*, page 34 and Figure 4. *UML Transliteration of SDAI EXPRESS Abstract Model, Part 2*, page 35, show the UML transliteration of the SDAI EXPRESS model for EXPRESS. This model was provided by David Price of PDIT, Inc. and IBM and carries the same caveats as the other models: it is an experiment in mapping and nothing more.

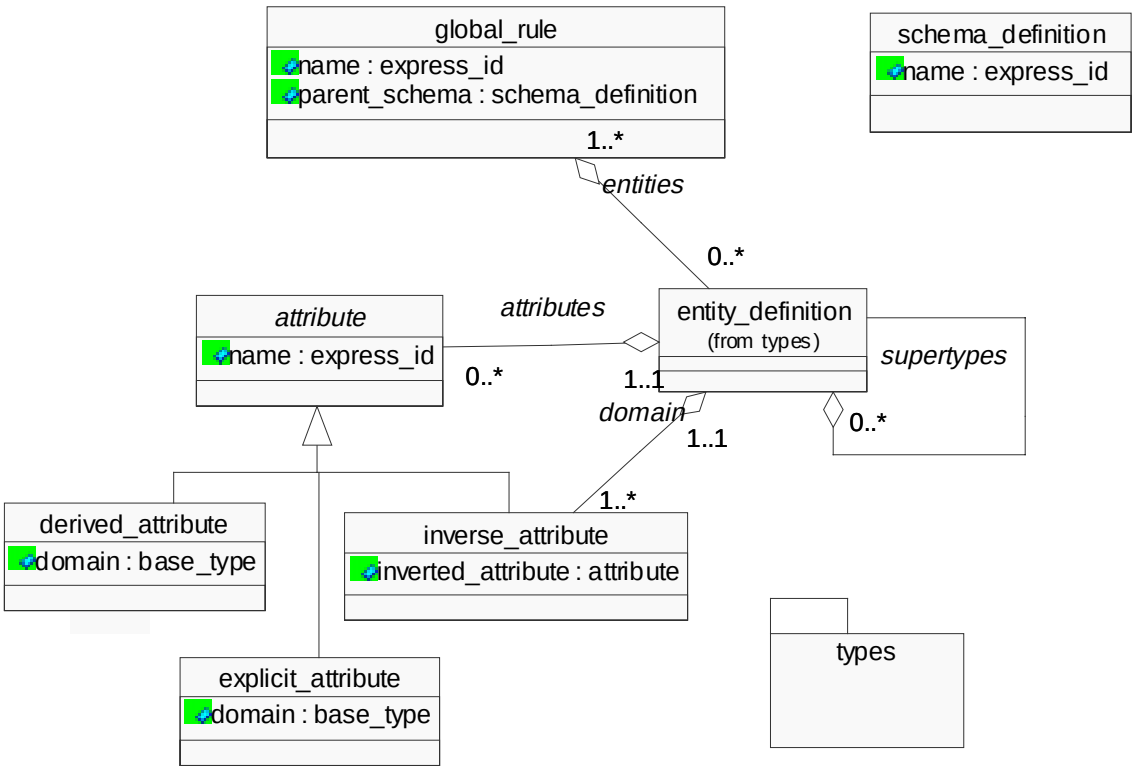

Figure 3 — UML Transliteration of SDAI EXPRESS Abstract Model, Part 1

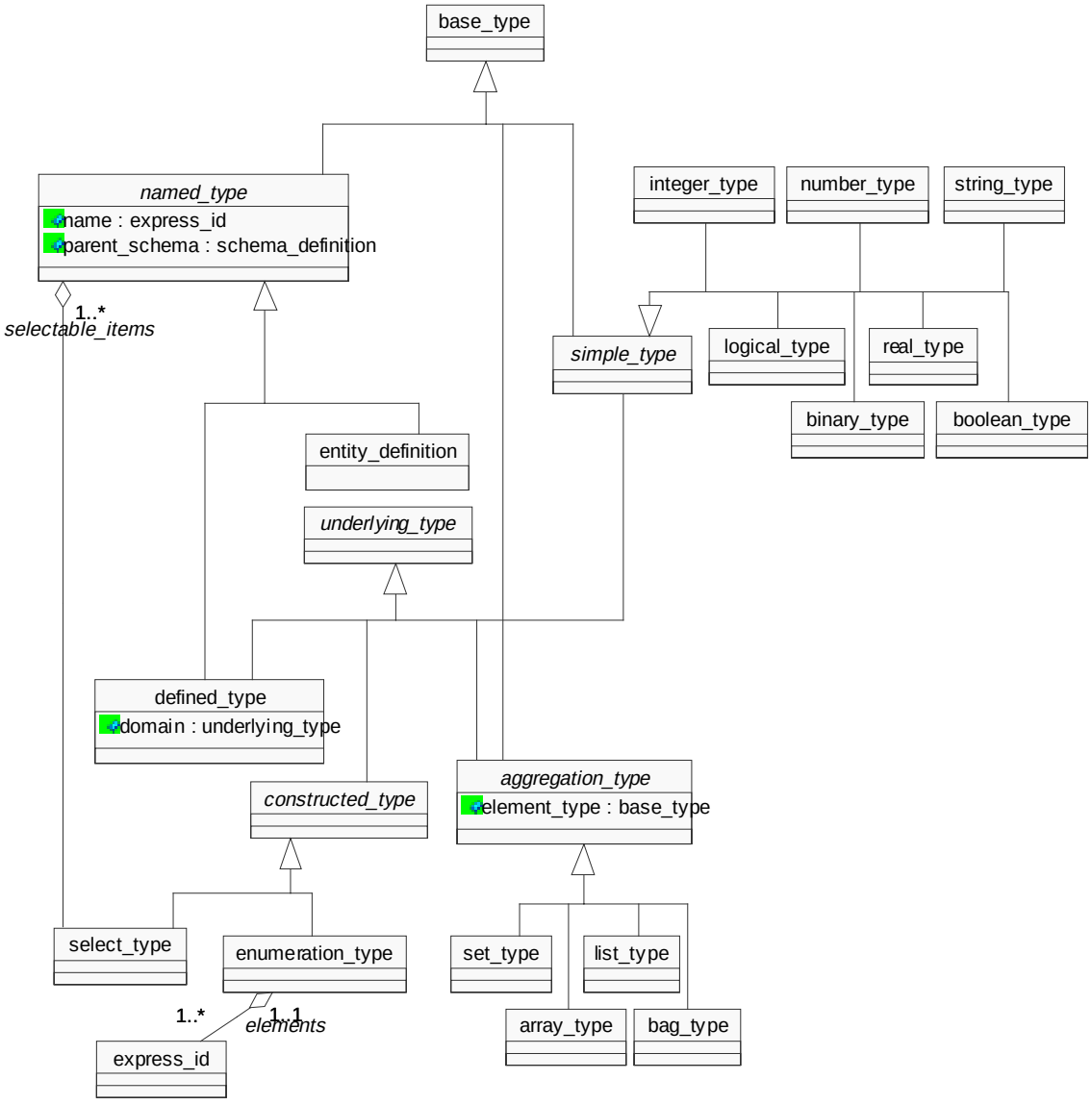

Figure 4 — UML Transliteration of SDAI EXPRESS Abstract Model, Part 2

3.4. UML Model from N228 EXPRESS Semantic Model

Figure 5. *UML Transliteration of EXPRESS Semantic Model from N228: overview*, page 36 (and subsequent figures), shows the UML transliteration of the EXPRESS semantic model for EXPRESS published in *TC184/SC4 N288*. This model was developed by the author according the mapping rules outlined above. Note that the EXPRESS semantic model published in N228 reflects the state of thinking at the time of publication, 1995. In private communication, Phil Spiby stressed that the state of thinking about EXPRESS abstractions has progressed significantly since then.

EXPRESS Schema Metamodel UML Transliteration
Version 0.1, 9 Oct 1999

From TC184/SC4 N228 by Phil Spiby, Eurostep, Uk, Ltd.

Adapted to UML by W. Eliot Kimber, ISOGEN International Corp.

This document is purely experimental and has absolutely no normative or official standing. It was developed for the express purpose of testing the ability to map EXPRESS into XMI using IBM's Rational Rose-to-XMI conversion tools.

This model adds to Phil's original EXPRESS metamodel a class for schema.

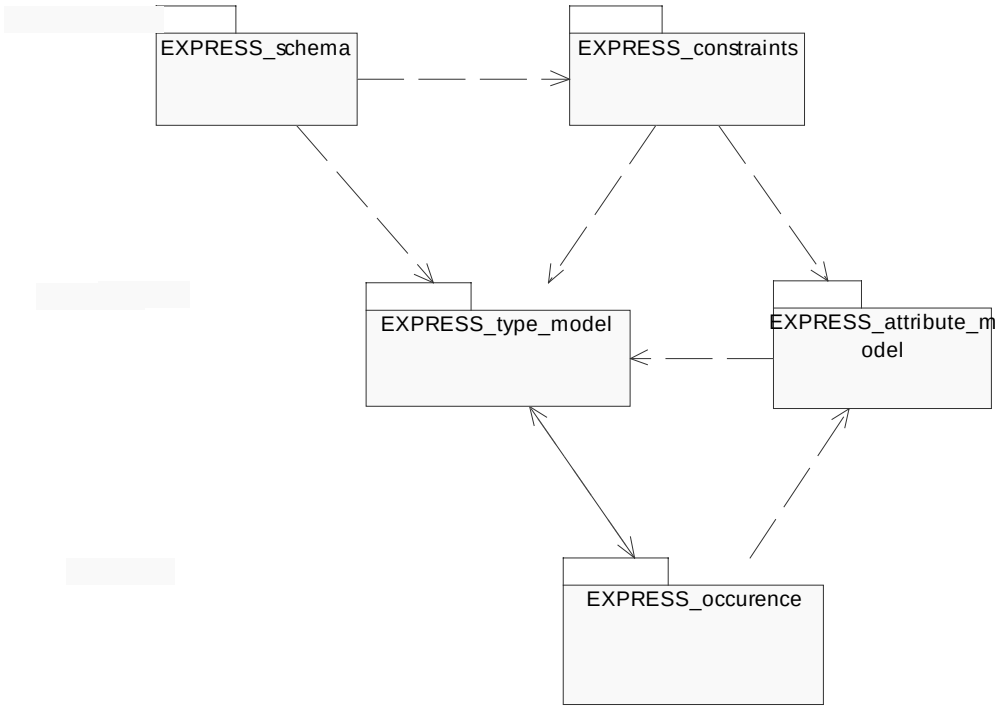

Figure 5 — UML Transliteration of EXPRESS Semantic Model from N228: overview

EXPRESS schema model

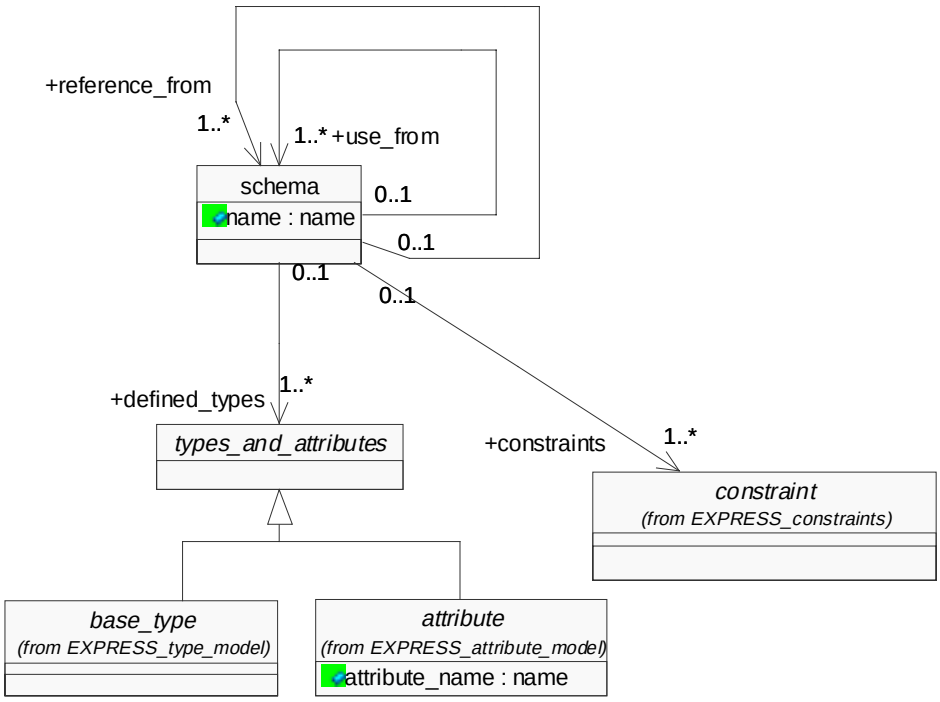

Figure 6 — UML Transliteration of EXPRESS Semantic Model from N228: EXPRESS schema package

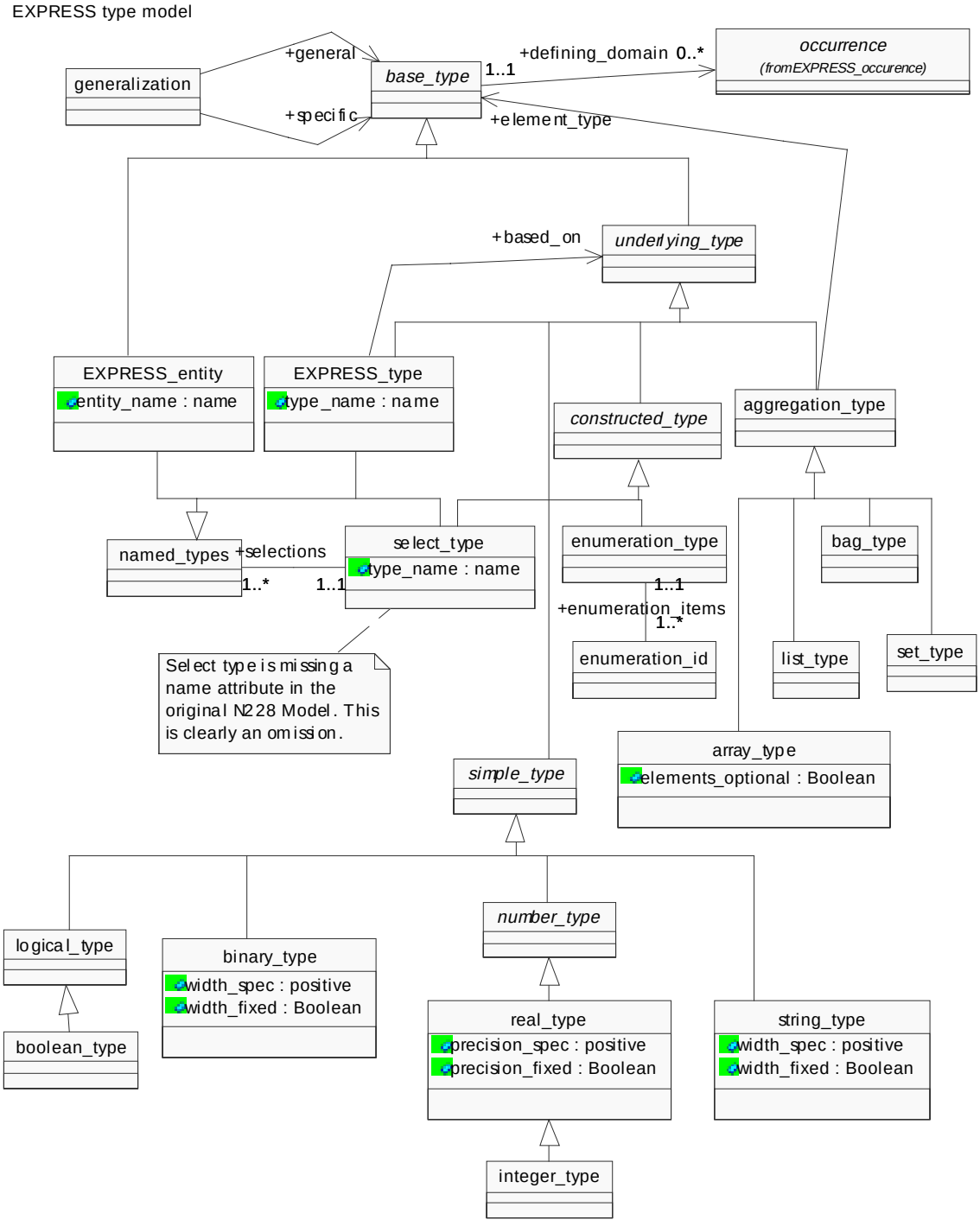

Figure 7 — UML Transliteration of EXPRESS Semantic Model from N228: EXPRESS type package

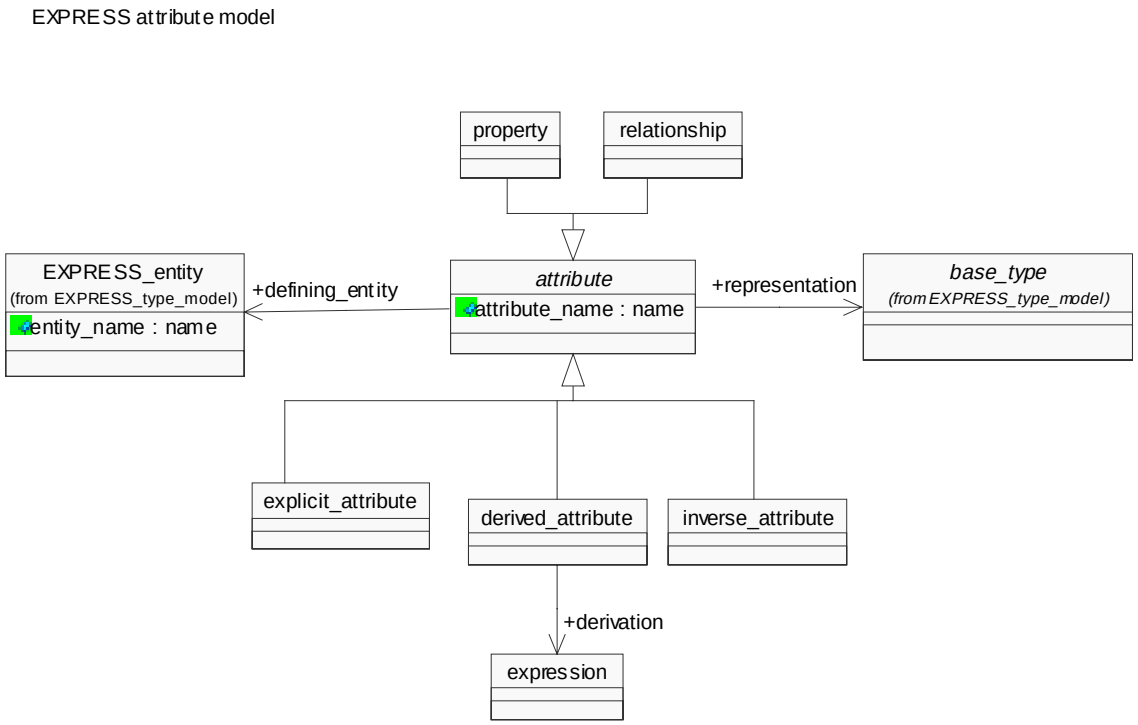

Figure 8 — UML Transliteration of EXPRESS Semantic Model from N228: EXPRESS attribute package

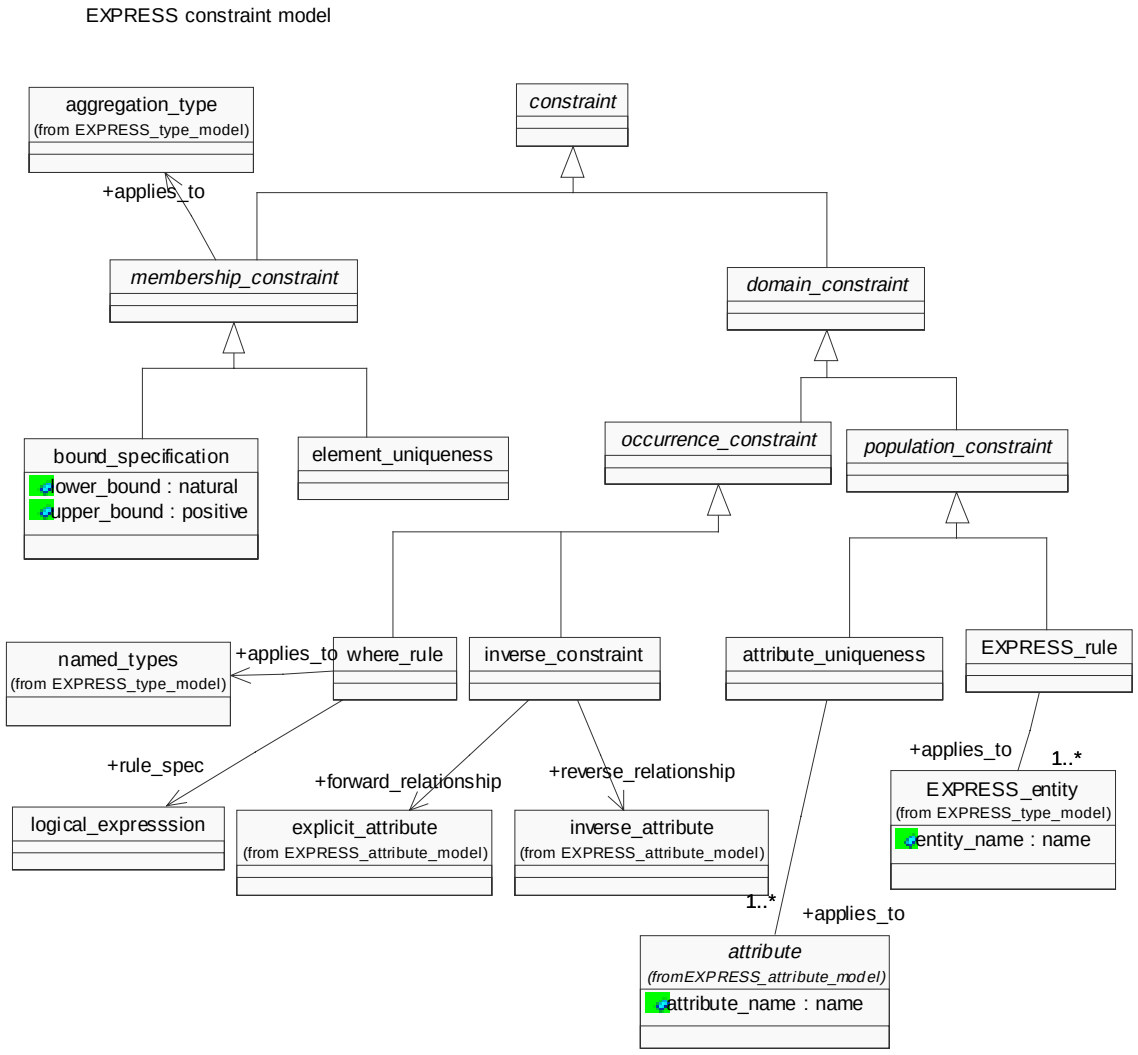

Figure 9 — UML Transliteration of EXPRESS Semantic Model from N228: Constraint package

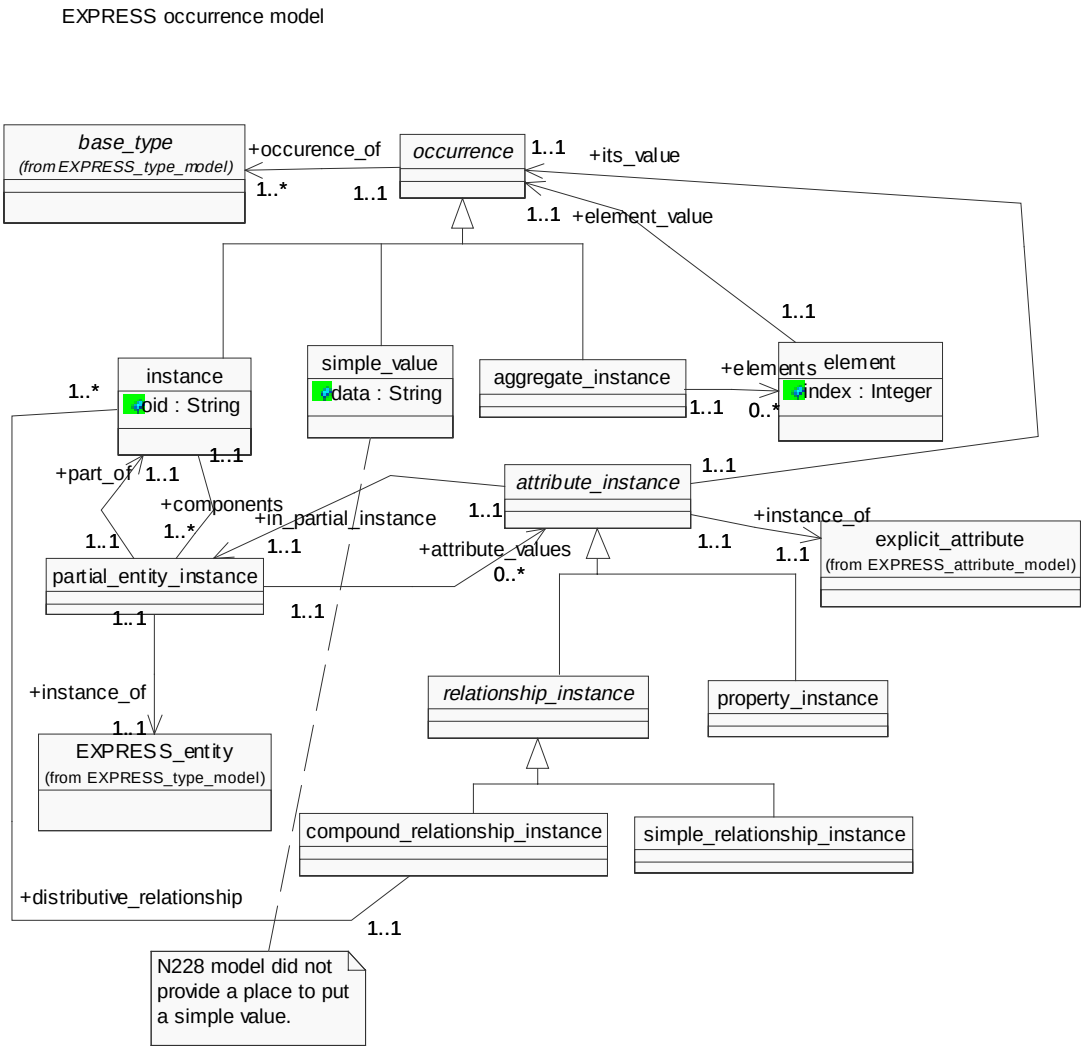

Figure 10 — UML Transliteration of EXPRESS Semantic Model from N228: Occurrence package

4. Evaluation of the Representation Methods

The three representation methods evaluated represent three fundamentally different design approaches for representing EXPRESS-driven data, reflecting different sets of requirements. The key requirements for the Part 28 Late Binding are completeness with respect to EXPRESS and generality, The key requirements for the PDML Early Binding are greatest representation of EXPRESS-defined constraints in the corresponding XML DTD and completeness with respect to EXPRESS. The key requirements for XMI are appropriateness for the representation of OMG-defined modeling languages and generality. These different requirements produce very different results. However, all three mechanisms are remarkably comparable.

Each of the three approaches appears to satisfy the requirement for completeness with respect to its governing standards or specifications. The key question appears to be the degree to which XMI can capture the semantic details of the representation of EXPRESS-driven data.

This section discusses the three representation mechanisms in detail:

- 4.1. *XML Metadata Interchange (XMI)*, page 42, discusses the OMG's XMI specification. This submission is described in the most detail
- 4.2. *Product Data Markup Language (PDIT)*, page 105, discusses the PDML Early Binding design, as refined through the two Early Binding workshops sponsored by the U.S. Department of Defense
- 4.3. *Part 28 Late Binding (ISO TC184/SC4)*, page 119, discusses the Part 28 Late Binding design

For the DTDs, the markup formatting style is that used by the authors of the declarations or that produced by the generating software. For instance data, a consistent markup style is used in which each start and end tag is on a separate line except for elements whose only content is a short text value. This makes it possible to judge the comparative verbosities of the different markup schemes by simple inspection. All the examples use the same people-and-pets schema and data instances.

4.1. XML Metadata Interchange (XMI)

The XML Metadata Interchange specification is a mechanism for defining XML document types and instances for the representation of metamodels and instances of those models. It's design is based on the OMG's Meta Object Facility (MOF), which along with the graphical Unified Modeling Language, forms the core of the OMG's metadata repository architecture. The XMI specification is a proposal submitted in response to a request for proposals for a "stream-based model interchange format" (SMIF). The SMIF RFP did not explicitly require the use of XML.

The MOF provides a layered system of abstraction that enables the mapping of data models to more general higher-level (more abstract) models. XMI defines a generic algorithm for mapping models or instances at any layer of abstraction into XML. Essentially, any model is simply an instance of the model one level of abstraction above it. By this same mechanism, instances of any model can be represented by applying the same algorithm in the other direction. This approach also provides different levels of early and late binding, with a built-in mapping between early and late bindings (through the mappings of the corresponding MOF models).

XMI was designed primarily to solve the problem of interchanging UML models, which currently have no defined canonical interchange representation. Because the MOF is very general, it is possible, at least in theory, to use XMI for any kind of model or instances if there is a UML model or metamodel. Note that XML is designed to enable the interchange of models and instances, although not necessarily in the same XML document.

XMI generation tools have been developed made publicly available by IBM and also incorporated into their VisualAge product.

4.1.1. Overview of the XMI Markup Approach

The XMI markup approach is defined as an algorithmic mapping between MOF constructs and XML element types and attributes that reflect them. The XMI specification defines two algorithms: one for generating XML DTD declarations for a model and one for generating XML instances. The DTDs generated are always early bound with respect to the models from which they are generated. Likewise, the XML instances are therefore early bound with respect to the instance objects they represent.

The XMI specification defines a set of constant element types that define a structural framework for an interchange package. It then defines what are essentially templates for the creation of model-specific element types. However, these templates are defined informally through prose and through algorithmic generation rules. They could have been defined as architectural forms in an SGML architecture

An XMI interchange document has a required header section with fixed element types for describing the interchange package itself, followed by the content to be interchanged. The content to be interchanged may be simply the XML representations of the instance objects or it may be a set of "difference" representations that describe changes to be applied to a previously-transmitted package. The XMI design also provides for "extensions", which are by definition non-standard parts of the package. The XMI-defined structures simply provide containers that isolate non-standard package components from standard package components.

The XMI header provides built-in element types for describing the package for tracking purposes: owner, contact, long and short descriptions, etc. These fields are comparable

to the fields provided by Part 21. The header may also contain pointers to the original models and metamodels.

The XMI markup uses the prefix "XMI." for all XMI-defined element types. Model-specific element types names are constructed by concatenating the model name with the class type name. Attributes are represented either by direct containment of the entities that are the value of the attribute or by reference to the entity that satisfies the value. In addition, XMI does not provide a generic mechanism for defining simple (non-entity) types. Instead, it uses the CORBA data types. This means that there is no obvious way to directly represent non-entity defined types from EXPRESS in the XMI created from the UML transliteration of the EXPRESS schema.

The MOF/UML-to-DTD generation rules are fairly simple, as outlined in section 6.3 of the XMI specification:

- All generated element type names are "fully qualified" through the concatenation of the name of each level of namespace (in the MOF sense), i.e., "package1.package2.class1.attribute1". Name components are separated with periods ".". All MOF names are also legal XML names, so there is no need to do any translation of characters in names (note that tools like Rational Rose do not appear to enforce the MOF name construction rules, for example, allowing spaces in attribute names).

- Each metamodel class is represented by an XML element type whose name is the class name (qualified by package name in a multi-package model). The content of the class-representing element contains elements that represent the class' properties, association ends, and directly-contained classes. Note that the properties, association ends, and contained classes are not separately grouped, so there is no obvious way to distinguish them in the instance data.

- Each class attribute is represented by an element type whose name is the attribute name, qualified by the class name. The attributes are allowed in their containing class' element in the order they are declared in the model. Note that associations translate into attributes that reflect the roles.

- Multiplicities are translated into repetition operators within content models. The mapping is as would be expected: 0..1 becomes "?", 1..1 requires no operator, 0..* becomes "*", 1..* becomes "+". All other multiplicities become "*". Note that in XML, multiplicities other than one or more or zero or more can be represented by writing out the content model (e.g., a multiplicity of 2..* for the role "a" generates a content model of (a,a+), which requires at least two "a" elements. However, this can lead to large content models in the extreme case (e.g., 100..10000) and so is not normally used for generated DTDs such as XMI and the PDML early binding).

- Attributes inherited from superclasses are "copied down" to the subclass instances. This is roughly analogous to partial entity instances in EXPRESS, but there is no explicit grouping of properties or associations by superclass. Rather, the required name qualification serves to disambiguate attributes from different superclasses.

- Attributes with simple types whose declared type is not one of the built-in CORBA types are represented by the content model "`(#PCDATA | XML.reference)*`", where content will be either some string representation of the value (details undefined by XMI) or a reference to the value.

  Attributes with enumerated or Boolean values are represented by elements that have an attribute whose declared value is a name group containing the possible enumeration values or "true" and "false" for Boolean values. (Note that the IBM XMI tools do not appear to apply this rule.)

- Each association role is represented by an element type whose name is the role name qualified with the class name of the class at the other end of the association. The association role element allows in its content the class that plays that role, as well as any of its subclasses.

- Each aggregation role is constructed as for association roles.

Because the XMI design uses containment of class instance elements in all contexts where they are used or referenced and provides no specialized per-class referencing element types, there would appear to be a problem in that the same class instance would need to be in multiple places at once (one instance for every association or aggregation role instance it was a member of). The XMI design solves this problem by allowing class elements to be hyperlinks and imposing constraints on the uses of those hyperlinks so that the instantiation and reference semantics will be properly reflected.

The MOF specifies that a given class instance can exist in no more than one containment (aggregation) relationship. Therefore, every class instance will have a well-defined place of primary existence within the class instance population: either as an independent object with no container or contained by one other object through an aggregation relationship. All other occurrences of the object instance should be pointers to the primary or "defining" instance. XMI represents these references by making the non-defining instances of the object into hyperlinks using the XLink syntax. These linking elements are called "proxies" in the XMI specification.

XMI allows the use of either direct ID references or XPointers for connecting proxies to defining elements. However, the XLink specification only recognizes the use of XPointers. Therefore, any proxy element that uses a direct ID reference will not be a conforming XLink.

In order to ensure that the links properly reflect the MOF and model-specific semantics, XMI imposes the following constraints on proxy elements:

- Proxy elements cannot have any content. That is, elements that are links cannot have local values for properties and associations as these will be defined by the defining element instance.

- Proxy elements must only link to elements of the same type. This is because the XMI semantics say that all instances of a given class that point to the same defining instance of that class represent a "union" defining a single object instance.
- The XMI spec says "proxies may be chained", which presumably means that a link from one proxy to another is equivalent to having the first proxy point directly to the defining element the second proxy points to. This seems to imply a multi-stage link resolution process. Note that the latest XLink working draft does not define any such indirect link resolution or addressing semantics, although it does not explicitly preclude them either.

Note that this use of hyperlinks as "proxies" corresponds exactly to the formal use-by-reference semantics defined by the value reference facility of the HyTime Standard (ISO/IEC 10744:1997), which is the ISO analog of the XLink/XPointer standards. The current XLink working draft does not provide as clear a use-by-reference semantic.

The rest of this section presents the XMI DTDs generated from the UML models shown above in order to discuss interesting aspects of the XMI design revealed by these samples.

NOTE: The DTD samples presented here were generated by the IBM XMI toolkit from UML models created using Rational Rose '98. They have not been rigorously checked against the actual XMI specification, in part because the XMI toolkit is represented as a reference implementation of the XMI mapping. This analysis assumes that these DTDs are an accurate reflection of the XMI spec (in any case, it seems unlikely that anyone would not use these tools to generate XMI given the complexity of XMI specification and the programming task involved).

The people and pets model is shown first as it is the simplest. Areas of interest are highlighted and labeled. The areas of interest are discussed following the presentation of the DTD source.

4.1.2. XMI DTD for People and Pets Model

This section presents the XMI DTD generated from the UML transliteration of the people and pets model. The common XMI-defined declarations have been omitted for brevity as they are the same for all generated XMI DTDs. Note that this DTD does not reflect a "pure" translation from the EXPRESS metamodel through the MOF mapping to an XMI early binding. Rather, it reflects a direct EXPRESS-to-UML mapping that may or may not be appropriate or accurate. However, lacking a normative mapping from EXPRESS constructs into UML constructs it is a reasonable demonstration of the type of early-bound markup that will result from applying the XMI rules to a UML transliteration of an EXPRESS model. Note that given a normative EXPRESS-to-UML mapping that the higher-level abstractions of the MOF are not really relevant to the problem of early-bound representation of EXPRESS data, as a normative EXPRESS-to-UML mapping would be a syntax-to-syntax mapping and could therefore stand on its own without the need to refer to a higher-level semantic mapping. However, the existence of a higher-level

semantic mapping would serve as a check on the correctness and completeness of the syntax-to-syntax mapping, which would otherwise be proven through inspection and testing.

Early-Bound XMI DTD

The following DTD represents the early binding of the people-and-pets model to XMI by generating an XMI DTD from the corresponding UML model.

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL:  people_and_pets                                          -->
<!-- ____________________________________________________________________ -->


<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  animal                                             -->
<!-- ____________________________________________________________________ -->


[1]<!ELEMENT animal.names (#PCDATA | XMI.reference)* >

[2]<!ELEMENT animal (animal.names?, XMI.extension*)? >
[3]<!ATTLIST animal
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  human                                              -->
<!-- ____________________________________________________________________ -->


<!ELEMENT human.pets (pet_animal | cat | dog)* >

[4]<!ELEMENT human.works_for (human)? >

<!ELEMENT human (animal.names?, XMI.extension*, human.pets*,
                 human.works_for?)? >
<!ATTLIST human
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  cat                                                -->
<!-- ____________________________________________________________________ -->


[5]<!ELEMENT cat.addicted_to_catnip (#PCDATA | XMI.reference)* >

<!ELEMENT cat (animal.names?, cat.addicted_to_catnip?,
               XMI.extension*)? >
<!ATTLIST cat
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
```

```
<!-- METAMODEL CLASS:  dog                                                  -->
<!-- _____________________________________________________________________ -->


<!ELEMENT dog.catches_frisbees (#PCDATA | XMI.reference)* >

<!ELEMENT dog (animal.names?, dog.catches_frisbees?, XMI.extension*)? >
<!ATTLIST dog
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _____________________________________________________________________ -->
<!--                                                                       -->
<!-- METAMODEL CLASS:  pet_animal                                          -->
<!-- _____________________________________________________________________ -->


[6]<!ELEMENT pet_animal (animal.names?, XMI.extension*)? >
<!ATTLIST pet_animal
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT people_and_pets ((animal | human | cat | dog | pet_animal)*) >
<!ATTLIST people_and_pets
            %XMI.element.att;
            %XMI.link.att;
>
            %XMI.element.att;
            %XMI.link.att;
>
```

Areas of interest in the people and pets DTD:

[1]

The attribute "names" for entity type "animal" is declared in the EXPRESS as a list of "name" type (which is derived from "string"). However, the XMI simply provides for the entering of either text (#PCDATA) or references to an "external value". Thus, the attribute value is not constrained in any way by the XML DTD, nor is the EXPRESS-defined data type reflected in the XMI data. This appears to be a side effect of the lack of a way to define named simple types in UML.

[2]

The "animal.names" element, which represents the attribute value is optional, but the model requires that all animals have names. This is because even though the model requires the attribute, the XMI mechanism of using class elements both for defining and referencing means that the entire content must be optional or it would not be possible to create "proxy" element instances that have no content, as the XMI rules require.

[3]

Note that there is nothing in the attribute list that indicates directly that this element type represents an instance of a class.

[4]

The `human.works_for` element represents the "works_for" attribute from the EXPRESS model. The element results from using the association label "works_for" in the UML association used to represent the EXPRESS attribute. Note that the content model allows a single human element. This correctly reflects the EXPRESS model that requires every human to work for exactly one other human (which may be themself).

In the instance data, the `human` element that goes inside the `human.works_for` element must be a "proxy" element that is a reference to the actual defining element.

[5]

According to the XMI mapping algorithm, this attribute should be represented by a binary element, rather than the generic attribute value content model used here. This appears to be a bug in the XMI Toolkit.

[6]

This element type, which is representing a UML class, results from the select type in the original EXPRESS model.

Early-Bound People and Pets Instance

Below is a sample early-bound instance conforming to the DTD declarations shown above, constructed from the same test data used in the examples. To construct this document, all entities are independent, in that there is no direct containment of "defining" elements and all references are represented using "proxy" elements. This is consistent with the general EXPRESS instantiation model and reflects the fact that aggregation is not used in the UML model at all. Note also that the value of the "animal.names" attribute uses a non-XML-defined and non-XMI-defined convention for representing a list of strings. This is an example of data representation that is unique to EXPRESS for which there is no satisfactory early-bound XMI representation. This same data is shown in a late-bound XMI instance using the EXPRESS meta-model DTD in the next section.

```
<?xml version="1.0"?>
<!DOCTYPE XMI SYSTEM "people_and_pets_xmi.dtd">
<XMI timestamp="19991011" verified="false">
<XMI.header>
<XMI.documentation><XMI.contact>W. Eliot Kimber</XMI.contact>
<XMI.shortDescription>People and pets sample
instance</XMI.shortDescription></XMI.documentation>
<XMI.model xmi.name="people_and_pets.exp"
xmi.version="0.3">people_and_pets.exp</XMI.model>
<XMI.metamodel
xmi.name="expr_sem_n228_uml.mdl">expr_sem_n228_uml.mdl</XMI.metamodel>
</XMI.header>
<XMI.content>
<people_and_pets>
<human xmi.id="oid001">
<animal.names>"W. Eliot Kimber"</animal.names>
<human.pets>
```

```
<dog href="#idref(oid005)" show="embed">
</dog>
<cat show="embed" href="#idref(oid003)"></cat>
<cat href="#idref(oid004)" show="embed"></cat>
</human.pets>
<human.works_for>
<human show="embed" href="#idref(oid002)"></human>
</human.works_for>
</human>
<human xmi.id="oid002">
<animal.names>"Renee Swank"</animal.names>
<human.works_for>
<human show="embed" href="#idref(oid006)"></human>
</human.works_for>
</human>
<cat xmi.id="oid003">
<animal.names>'Sigfried'</animal.names>
<cat.addicted_to_catnip>True</cat.addicted_to_catnip>
</cat>
<cat xmi.id="oid004">
<animal.names>'Bete Noir', 'Beteski'</animal.names>
<cat.addicted_to_catnip>True</cat.addicted_to_catnip>
</cat>
<dog xmi.id="oid005">
<animal.names>'Forrest', 'Dogboy', 'Noseboy'</animal.names>
<dog.catches_frisbees>false</dog.catches_frisbees>
</dog>
<human xmi.id="oid006">
<animal.names>'Carla Corkern', 'Task Mistress'</animal.names>
<human.pets>
<dog href="#idref(oid008)"></dog>
<dog href="#idref(oid009)"></dog>
</human.pets>
<human.works_for>
<human show="embed" href="#idref(oid007)"/>
</human.works_for>
</human>
<human xmi.id="oid007">
<animal.names>'Lucie Feldstat'</animal.names>
<human.works_for>
<human show="embed" href="#idref(oid007)"></human>
</human.works_for>
</human>
<dog xmi.id="oid008">
<animal.names>'Chauncy', 'The Chaunce Man'</animal.names>
<dog.catches_frisbees>false</dog.catches_frisbees>
</dog>
<dog xmi.id="oid009">
<animal.names>'Chelsie', 'The Chelster'</animal.names>
<dog.catches_frisbees>false</dog.catches_frisbees>
</dog>
</people_and_pets>
</XMI.content>
</XMI>
```

4.1.3. XMI Interchange Using EXPRESS Semantic Model

This section presents the XMI DTD generated from the UML transliteration of the EXPRESS semantic model. This DTD allows the creation of both XMI representations of EXPRESS schemas and late-bound entity instance populations.

XMI DTD for EXPRESS Semantic Model

This DTD reflects the UML transliteration of the EXPRESS meta model as defined in *N228*. Because the metamodel includes occurrences as well as schema components, it can be used to represent entity instances as well as schemas. Note that with respect to the schema this DTD is early bound, that is, the XMI elements directly reflect the classes that make up an EXPRESS schema, but that for the data instances, it is late bound. The corresponding data instance is presented in the next section.

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL:  expr_sem_n228                                            -->
<!-- ____________________________________________________________________ -->

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL PACKAGE:  EXPRESS_schema                                   -->
<!-- ____________________________________________________________________ -->

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  schema                                             -->
<!-- ____________________________________________________________________ -->

<!ELEMENT EXPRESS_schema.schema.name (#PCDATA | XMI.reference)* >

<!ELEMENT EXPRESS_schema.schema.use_from (EXPRESS_schema.schema)* >

<!ELEMENT EXPRESS_schema.schema.reference_from
                                                (EXPRESS_schema.schema)*
>

<!ELEMENT EXPRESS_schema.schema.defined_types
(EXPRESS_schema.types_and_attributes |

EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |
```

```
EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity |

EXPRESS_attribute_model.attribute |

EXPRESS_attribute_model.explicit_attribute |

EXPRESS_attribute_model.derived_attribute |

EXPRESS_attribute_model.inverse_attribute |

EXPRESS_attribute_model.relationship |

EXPRESS_attribute_model.property)*
                                                >

<!ELEMENT EXPRESS_schema.schema.constraints
(EXPRESS_constraints.constraint |

EXPRESS_constraints.domain_constraint |

EXPRESS_constraints.occurrence_constraint |

EXPRESS_constraints.inverse_constraint |

EXPRESS_constraints.where_rule |

EXPRESS_constraints.population_constraint |

EXPRESS_constraints.attribute_uniqueness |

EXPRESS_constraints.EXPRESS_rule |

EXPRESS_constraints.membership_constraint |

EXPRESS_constraints.bound_specification |

EXPRESS_constraints.element_uniqueness)*
                                              >

<!ELEMENT EXPRESS_schema.schema (EXPRESS_schema.schema.name?,
                                 XMI.extension*,
                                 EXPRESS_schema.schema.use_from*,
                                 EXPRESS_schema.schema.reference_from*,
                                 EXPRESS_schema.schema.defined_types*,
                                 EXPRESS_schema.schema.constraints*)? >
<!ATTLIST EXPRESS_schema.schema
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  types_and_attributes                               -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_schema.types_and_attributes (XMI.extension*)? >
<!ATTLIST EXPRESS_schema.types_and_attributes
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT EXPRESS_schema ((EXPRESS_schema.schema |
                           EXPRESS_schema.types_and_attributes)*) >
<!ATTLIST EXPRESS_schema
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL PACKAGE:  EXPRESS_type_model                               -->
<!-- ____________________________________________________________________ -->


<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  generalization                                     -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.generalization.specific
(EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |
```

```
EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                        >

<!ELEMENT EXPRESS_type_model.generalization.general
(EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                      >

<!ELEMENT EXPRESS_type_model.generalization (XMI.extension*,

EXPRESS_type_model.generalization.specific*,

EXPRESS_type_model.generalization.general*)?
                                                    >
<!ATTLIST EXPRESS_type_model.generalization
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
```

```
<!--                                                                    -->
<!-- METAMODEL CLASS:  base_type                                        -->
<!-- ______________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.base_type.defining_domain
(EXPRESS_occurence.occurrence |

EXPRESS_occurence.instance |

EXPRESS_occurence.aggregate_instance |

EXPRESS_occurence.simple_value)*
                                                          >

<!ELEMENT EXPRESS_type_model.base_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                        >
<!ATTLIST EXPRESS_type_model.base_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  underlying_type                              -->
<!-- ______________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.underlying_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                             >
<!ATTLIST EXPRESS_type_model.underlying_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  EXPRESS_entity                               -->
<!-- ______________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.EXPRESS_entity.entity_name (#PCDATA |
                                                          XMI.reference)*
>

<!ELEMENT EXPRESS_type_model.EXPRESS_entity.eXPRESS_rule

(EXPRESS_constraints.EXPRESS_rule)* >

<!ELEMENT EXPRESS_type_model.EXPRESS_entity
(EXPRESS_type_model.EXPRESS_entity.entity_name?,
                                          XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.named_types.select_type?,

EXPRESS_type_model.EXPRESS_entity.eXPRESS_rule*)?
                                           >
<!ATTLIST EXPRESS_type_model.EXPRESS_entity
```

```
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  EXPRESS_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.EXPRESS_type.type_name (#PCDATA |
                                                     XMI.reference)* >

<!ELEMENT EXPRESS_type_model.EXPRESS_type.based_on
(EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type)*
                                                    >

<!ELEMENT EXPRESS_type_model.EXPRESS_type
(EXPRESS_type_model.EXPRESS_type.type_name?,
                                          XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.named_types.select_type?,

EXPRESS_type_model.EXPRESS_type.based_on*)?
                                            >
<!ATTLIST EXPRESS_type_model.EXPRESS_type
            %XMI.element.att;
            %XMI.link.att;
```

```
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  constructed_type                                   -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.constructed_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                                    >
<!ATTLIST EXPRESS_type_model.constructed_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  named_types                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.named_types.select_type

(EXPRESS_type_model.select_type)? >

<!ELEMENT EXPRESS_type_model.named_types (XMI.extension*,

EXPRESS_type_model.named_types.select_type?)?
                                              >
<!ATTLIST EXPRESS_type_model.named_types
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  select_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.select_type.type_name (#PCDATA |
                                                    XMI.reference)* >

<!ELEMENT EXPRESS_type_model.select_type.selections
(EXPRESS_type_model.named_types |

EXPRESS_type_model.select_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                       >

<!ELEMENT EXPRESS_type_model.select_type
(EXPRESS_type_model.select_type.type_name?,
                                          XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.named_types.select_type?,

EXPRESS_type_model.select_type.selections*)?
```

```
                                                >
<!ATTLIST EXPRESS_type_model.select_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  enumeration_type                                   -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.enumeration_type.enumeration_items

(EXPRESS_type_model.enumeration_id)* >

<!ELEMENT EXPRESS_type_model.enumeration_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.enumeration_type.enumeration_items*)?
                                                    >
<!ATTLIST EXPRESS_type_model.enumeration_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  enumeration_id                                     -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.enumeration_id.enumeration_type

(EXPRESS_type_model.enumeration_type)? >

<!ELEMENT EXPRESS_type_model.enumeration_id (XMI.extension*,

EXPRESS_type_model.enumeration_id.enumeration_type?)?
                                                  >
<!ATTLIST EXPRESS_type_model.enumeration_id
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  array_type                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.array_type.elements_optional (#PCDATA |

XMI.reference)*
                                                              >

<!ELEMENT EXPRESS_type_model.array_type
(EXPRESS_type_model.array_type.elements_optional?,
                                          XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.aggregation_type.element_type*)?
```

```
                                                >
<!ATTLIST EXPRESS_type_model.array_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  aggregation_type                                   -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.aggregation_type.element_type
(EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                              >

<!ELEMENT EXPRESS_type_model.aggregation_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.aggregation_type.element_type*)?
                                                    >
<!ATTLIST EXPRESS_type_model.aggregation_type
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  bag_type                                           -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.bag_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.aggregation_type.element_type*)?
                                          >
<!ATTLIST EXPRESS_type_model.bag_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  list_type                                          -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.list_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.aggregation_type.element_type*)?
                                           >
<!ATTLIST EXPRESS_type_model.list_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  set_type                                           -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.set_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*,

EXPRESS_type_model.aggregation_type.element_type*)?
                                          >
<!ATTLIST EXPRESS_type_model.set_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  simple_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.simple_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                            >
<!ATTLIST EXPRESS_type_model.simple_type
            %XMI.element.att;
```

```
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  logical_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.logical_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                              >
<!ATTLIST EXPRESS_type_model.logical_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  boolean_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.boolean_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                               >
<!ATTLIST EXPRESS_type_model.boolean_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  binary_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.binary_type.width_spec (#PCDATA |
                                                     XMI.reference)* >

<!ELEMENT EXPRESS_type_model.binary_type.width_fixed (#PCDATA |
                                                      XMI.reference)* >

<!ELEMENT EXPRESS_type_model.binary_type
(EXPRESS_type_model.binary_type.width_spec?,

EXPRESS_type_model.binary_type.width_fixed?,
                                           XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                             >
<!ATTLIST EXPRESS_type_model.binary_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  number_type                                        -->
<!-- ____________________________________________________________________ -->
```

```
<!ELEMENT EXPRESS_type_model.number_type (XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                          >
<!ATTLIST EXPRESS_type_model.number_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  real_type                                          -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.real_type.precision_spec (#PCDATA |
                                                       XMI.reference)* >

<!ELEMENT EXPRESS_type_model.real_type.precision_fixed (#PCDATA |
                                                        XMI.reference)* >

<!ELEMENT EXPRESS_type_model.real_type
(EXPRESS_type_model.real_type.precision_spec?,

EXPRESS_type_model.real_type.precision_fixed?,
                                        XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                         >
<!ATTLIST EXPRESS_type_model.real_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  integer_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.integer_type
(EXPRESS_type_model.real_type.precision_spec?,

EXPRESS_type_model.real_type.precision_fixed?,
                                          XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                           >
<!ATTLIST EXPRESS_type_model.integer_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  string_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_type_model.string_type.width_spec (#PCDATA |
                                                     XMI.reference)* >

<!ELEMENT EXPRESS_type_model.string_type.width_fixed (#PCDATA |
                                                      XMI.reference)* >
```

```
<!ELEMENT EXPRESS_type_model.string_type
(EXPRESS_type_model.string_type.width_spec?,

EXPRESS_type_model.string_type.width_fixed?,
                                        XMI.extension*,

EXPRESS_type_model.base_type.defining_domain*)?
                                         >
<!ATTLIST EXPRESS_type_model.string_type
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT EXPRESS_type_model ((EXPRESS_type_model.generalization |
                               EXPRESS_type_model.base_type |
                               EXPRESS_type_model.underlying_type |
                               EXPRESS_type_model.EXPRESS_entity |
                               EXPRESS_type_model.EXPRESS_type |
                               EXPRESS_type_model.constructed_type |
                               EXPRESS_type_model.named_types |
                               EXPRESS_type_model.select_type |
                               EXPRESS_type_model.enumeration_type |
                               EXPRESS_type_model.enumeration_id |
                               EXPRESS_type_model.array_type |
                               EXPRESS_type_model.aggregation_type |
                               EXPRESS_type_model.bag_type |
                               EXPRESS_type_model.list_type |
                               EXPRESS_type_model.set_type |
                               EXPRESS_type_model.simple_type |
                               EXPRESS_type_model.logical_type |
                               EXPRESS_type_model.boolean_type |
                               EXPRESS_type_model.binary_type |
                               EXPRESS_type_model.number_type |
                               EXPRESS_type_model.real_type |
                               EXPRESS_type_model.integer_type |
                               EXPRESS_type_model.string_type)*) >
<!ATTLIST EXPRESS_type_model
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL PACKAGE:  EXPRESS_constraints                              -->
<!-- ____________________________________________________________________ -->


<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  constraint                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.constraint (XMI.extension*)? >
<!ATTLIST EXPRESS_constraints.constraint
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  logical_expression                                -->
<!-- ____________________________________________________________________ -->
```

```
<!ELEMENT EXPRESS_constraints.logical_expression (XMI.extension*)? >
<!ATTLIST EXPRESS_constraints.logical_expression
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  where_rule                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.where_rule.applies_to
(EXPRESS_type_model.named_types |

EXPRESS_type_model.select_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                        >

<!ELEMENT EXPRESS_constraints.where_rule.rule_spec

(EXPRESS_constraints.logical_expression)* >

<!ELEMENT EXPRESS_constraints.where_rule (XMI.extension*,

EXPRESS_constraints.where_rule.applies_to*,

EXPRESS_constraints.where_rule.rule_spec*)?
                                              >
<!ATTLIST EXPRESS_constraints.where_rule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  inverse_constraint                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.inverse_constraint.forward_relationship

(EXPRESS_attribute_model.explicit_attribute)* >

<!ELEMENT EXPRESS_constraints.inverse_constraint.reverse_relationship

(EXPRESS_attribute_model.inverse_attribute)* >

<!ELEMENT EXPRESS_constraints.inverse_constraint (XMI.extension*,

EXPRESS_constraints.inverse_constraint.forward_relationship*,

EXPRESS_constraints.inverse_constraint.reverse_relationship*)?
                                                    >
<!ATTLIST EXPRESS_constraints.inverse_constraint
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  attribute_uniqueness                               -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.attribute_uniqueness.applies_to
(EXPRESS_attribute_model.attribute |

EXPRESS_attribute_model.explicit_attribute |

EXPRESS_attribute_model.derived_attribute |

EXPRESS_attribute_model.inverse_attribute |

EXPRESS_attribute_model.relationship |

EXPRESS_attribute_model.property)*
                                                                    >

<!ELEMENT EXPRESS_constraints.attribute_uniqueness (XMI.extension*,

EXPRESS_constraints.attribute_uniqueness.applies_to*)?
                                                          >
<!ATTLIST EXPRESS_constraints.attribute_uniqueness
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  EXPRESS_rule                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.EXPRESS_rule.applies_to

(EXPRESS_type_model.EXPRESS_entity)* >

<!ELEMENT EXPRESS_constraints.EXPRESS_rule (XMI.extension*,

EXPRESS_constraints.EXPRESS_rule.applies_to*)?
                                              >
<!ATTLIST EXPRESS_constraints.EXPRESS_rule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  population_constraint                              -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.population_constraint (XMI.extension*)? >
<!ATTLIST EXPRESS_constraints.population_constraint
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  occurrence_constraint                              -->
<!-- ____________________________________________________________________ -->
```

```
<!ELEMENT EXPRESS_constraints.occurrence_constraint (XMI.extension*)? >
<!ATTLIST EXPRESS_constraints.occurrence_constraint
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  domain_constraint                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.domain_constraint (XMI.extension*)? >
<!ATTLIST EXPRESS_constraints.domain_constraint
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  membership_constraint                              -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.membership_constraint.applies_to
(EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type)*
                                                                    >

<!ELEMENT EXPRESS_constraints.membership_constraint (XMI.extension*,

EXPRESS_constraints.membership_constraint.applies_to*)?
                                                      >
<!ATTLIST EXPRESS_constraints.membership_constraint
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  bound_specification                                -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.bound_specification.lower_bound (#PCDATA |

XMI.reference)*
                                                                  >

<!ELEMENT EXPRESS_constraints.bound_specification.upper_bound (#PCDATA |

XMI.reference)*
                                                                  >
```

```
<!ELEMENT EXPRESS_constraints.bound_specification
(EXPRESS_constraints.bound_specification.lower_bound?,

EXPRESS_constraints.bound_specification.upper_bound?,
                                                XMI.extension*,

EXPRESS_constraints.membership_constraint.applies_to*)?
                                                 >
<!ATTLIST EXPRESS_constraints.bound_specification
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  element_uniqueness                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_constraints.element_uniqueness (XMI.extension*,

EXPRESS_constraints.membership_constraint.applies_to*)?
                                                 >
<!ATTLIST EXPRESS_constraints.element_uniqueness
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT EXPRESS_constraints ((EXPRESS_constraints.constraint |
                                EXPRESS_constraints.logical_expression |
                                EXPRESS_constraints.where_rule |
                                EXPRESS_constraints.inverse_constraint |
                                EXPRESS_constraints.attribute_uniqueness |
                                EXPRESS_constraints.EXPRESS_rule |
                                EXPRESS_constraints.population_constraint
|
                                EXPRESS_constraints.occurrence_constraint
|
                                EXPRESS_constraints.domain_constraint |
                                EXPRESS_constraints.membership_constraint
|
                                EXPRESS_constraints.bound_specification |
                                EXPRESS_constraints.element_uniqueness)*)
>
<!ATTLIST EXPRESS_constraints
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL PACKAGE:  EXPRESS_attribute_model                          -->
<!-- ____________________________________________________________________ -->


<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  property                                           -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.property
(EXPRESS_attribute_model.attribute.attribute_name?,
                                            XMI.extension*,
```

```
EXPRESS_attribute_model.attribute.defining_entity*,

EXPRESS_attribute_model.attribute.representation*,

EXPRESS_attribute_model.attribute.attribute_uniqueness*)?
                                                    >
<!ATTLIST EXPRESS_attribute_model.property
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  relationship                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.relationship
(EXPRESS_attribute_model.attribute.attribute_name?,
                                                XMI.extension*,

EXPRESS_attribute_model.attribute.defining_entity*,

EXPRESS_attribute_model.attribute.representation*,

EXPRESS_attribute_model.attribute.attribute_uniqueness*)?
                                                     >
<!ATTLIST EXPRESS_attribute_model.relationship
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  attribute                                          -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.attribute.attribute_name (#PCDATA |

XMI.reference)*
                                                                  >

<!ELEMENT EXPRESS_attribute_model.attribute.defining_entity

(EXPRESS_type_model.EXPRESS_entity)* >

<!ELEMENT EXPRESS_attribute_model.attribute.representation
(EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |
```

```
EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                        >

<!ELEMENT EXPRESS_attribute_model.attribute.attribute_uniqueness

(EXPRESS_constraints.attribute_uniqueness)* >

<!ELEMENT EXPRESS_attribute_model.attribute
(EXPRESS_attribute_model.attribute.attribute_name?,
                                            XMI.extension*,

EXPRESS_attribute_model.attribute.defining_entity*,

EXPRESS_attribute_model.attribute.representation*,

EXPRESS_attribute_model.attribute.attribute_uniqueness*)?
                                             >
<!ATTLIST EXPRESS_attribute_model.attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  explicit_attribute                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.explicit_attribute
(EXPRESS_attribute_model.attribute.attribute_name?,
                                                    XMI.extension*,

EXPRESS_attribute_model.attribute.defining_entity*,

EXPRESS_attribute_model.attribute.representation*,

EXPRESS_attribute_model.attribute.attribute_uniqueness*)?
                                                     >
<!ATTLIST EXPRESS_attribute_model.explicit_attribute
            %XMI.element.att;
            %XMI.link.att;
```

```
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  derived_attribute                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.derived_attribute.derivation

(EXPRESS_attribute_model.expression)* >

<!ELEMENT EXPRESS_attribute_model.derived_attribute
(EXPRESS_attribute_model.attribute.attribute_name?,
                                                    XMI.extension*,

EXPRESS_attribute_model.attribute.defining_entity*,

EXPRESS_attribute_model.attribute.representation*,

EXPRESS_attribute_model.attribute.attribute_uniqueness*,

EXPRESS_attribute_model.derived_attribute.derivation*)?
                                                     >
<!ATTLIST EXPRESS_attribute_model.derived_attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  inverse_attribute                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.inverse_attribute
(EXPRESS_attribute_model.attribute.attribute_name?,
                                                    XMI.extension*,

EXPRESS_attribute_model.attribute.defining_entity*,

EXPRESS_attribute_model.attribute.representation*,

EXPRESS_attribute_model.attribute.attribute_uniqueness*)?
                                                     >
<!ATTLIST EXPRESS_attribute_model.inverse_attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  expression                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_attribute_model.expression (XMI.extension*)? >
<!ATTLIST EXPRESS_attribute_model.expression
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT EXPRESS_attribute_model ((EXPRESS_attribute_model.property |
                                    EXPRESS_attribute_model.relationship |
```

```
                                    EXPRESS_attribute_model.attribute |

EXPRESS_attribute_model.explicit_attribute |

EXPRESS_attribute_model.derived_attribute |

EXPRESS_attribute_model.inverse_attribute |
                                    EXPRESS_attribute_model.expression)*)
>
<!ATTLIST EXPRESS_attribute_model
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL PACKAGE:  EXPRESS_occurence                                -->
<!-- ____________________________________________________________________ -->


<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  instance                                           -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.instance.oid (#PCDATA | XMI.reference)* >

<!ELEMENT EXPRESS_occurence.instance.components

(EXPRESS_occurence.partial_entity_instance)* >

<!ELEMENT EXPRESS_occurence.instance.compound_relationship_instance

(EXPRESS_occurence.compound_relationship_instance)? >

<!ELEMENT EXPRESS_occurence.instance (EXPRESS_occurence.instance.oid?,
                                      XMI.extension*,

EXPRESS_occurence.occurrence.base_type*,

EXPRESS_occurence.occurrence.occurence_of*,

EXPRESS_occurence.instance.components*,

EXPRESS_occurence.instance.compound_relationship_instance?)?
                                       >
<!ATTLIST EXPRESS_occurence.instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  simple_value                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.simple_value.data (#PCDATA |
                                               XMI.reference)* >

<!ELEMENT EXPRESS_occurence.simple_value
(EXPRESS_occurence.simple_value.data?,
                                         XMI.extension*,
```

```
EXPRESS_occurence.occurrence.base_type*,

EXPRESS_occurence.occurrence.occurence_of*)?
                                                >
<!ATTLIST EXPRESS_occurence.simple_value
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  aggregate_instance                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.aggregate_instance.elements

(EXPRESS_occurence.element)* >

<!ELEMENT EXPRESS_occurence.aggregate_instance (XMI.extension*,

EXPRESS_occurence.occurrence.base_type*,

EXPRESS_occurence.occurrence.occurence_of*,

EXPRESS_occurence.aggregate_instance.elements*)?
                                                      >
<!ATTLIST EXPRESS_occurence.aggregate_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  element                                            -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.element.index (#PCDATA | XMI.reference)* >

<!ELEMENT EXPRESS_occurence.element.element_value
(EXPRESS_occurence.occurrence |

EXPRESS_occurence.instance |

EXPRESS_occurence.aggregate_instance |

EXPRESS_occurence.simple_value)?
                                                    >

<!ELEMENT EXPRESS_occurence.element (EXPRESS_occurence.element.index?,
                                     XMI.extension*,

EXPRESS_occurence.element.element_value?)?
                                      >
<!ATTLIST EXPRESS_occurence.element
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  partial_entity_instance                            -->
```

```
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.partial_entity_instance.part_of

(EXPRESS_occurence.instance)? >

<!ELEMENT EXPRESS_occurence.partial_entity_instance.instance

(EXPRESS_occurence.instance)? >

<!ELEMENT EXPRESS_occurence.partial_entity_instance.attribute_values
(EXPRESS_occurence.attribute_instance |

EXPRESS_occurence.relationship_instance |

EXPRESS_occurence.compound_relationship_instance |

EXPRESS_occurence.simple_relationship_instance |

EXPRESS_occurence.property_instance)*
                                                                      >

<!ELEMENT EXPRESS_occurence.partial_entity_instance.instance_of

(EXPRESS_type_model.EXPRESS_entity)? >

<!ELEMENT EXPRESS_occurence.partial_entity_instance (XMI.extension*,

EXPRESS_occurence.partial_entity_instance.part_of?,

EXPRESS_occurence.partial_entity_instance.instance?,

EXPRESS_occurence.partial_entity_instance.attribute_values*,

EXPRESS_occurence.partial_entity_instance.instance_of?)?
                                                    >
<!ATTLIST EXPRESS_occurence.partial_entity_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  attribute_instance                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.attribute_instance.instance_of

(EXPRESS_attribute_model.explicit_attribute)? >

<!ELEMENT EXPRESS_occurence.attribute_instance.its_value
(EXPRESS_occurence.occurrence |

EXPRESS_occurence.instance |

EXPRESS_occurence.aggregate_instance |

EXPRESS_occurence.simple_value)?
                                                          >

<!ELEMENT EXPRESS_occurence.attribute_instance.in_partial_instance
```

```
(EXPRESS_occurence.partial_entity_instance)? >

<!ELEMENT EXPRESS_occurence.attribute_instance (XMI.extension*,

EXPRESS_occurence.attribute_instance.instance_of?,

EXPRESS_occurence.attribute_instance.its_value?,

EXPRESS_occurence.attribute_instance.in_partial_instance?)?
                                                    >
<!ATTLIST EXPRESS_occurence.attribute_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  relationship_instance                              -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.relationship_instance (XMI.extension*,

EXPRESS_occurence.attribute_instance.instance_of?,

EXPRESS_occurence.attribute_instance.its_value?,

EXPRESS_occurence.attribute_instance.in_partial_instance?)?
                                                      >
<!ATTLIST EXPRESS_occurence.relationship_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  property_instance                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.property_instance (XMI.extension*,

EXPRESS_occurence.attribute_instance.instance_of?,

EXPRESS_occurence.attribute_instance.its_value?,

EXPRESS_occurence.attribute_instance.in_partial_instance?)?
                                                    >
<!ATTLIST EXPRESS_occurence.property_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  compound_relationship_instance                     -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.compound_relationship_instance.instance

(EXPRESS_occurence.instance)* >
```

```
<!ELEMENT
EXPRESS_occurence.compound_relationship_instance.distributive_relationshi
p

(EXPRESS_occurence.instance)* >

<!ELEMENT EXPRESS_occurence.compound_relationship_instance
(XMI.extension*,

EXPRESS_occurence.attribute_instance.instance_of?,

EXPRESS_occurence.attribute_instance.its_value?,

EXPRESS_occurence.attribute_instance.in_partial_instance?,

EXPRESS_occurence.compound_relationship_instance.instance*,

EXPRESS_occurence.compound_relationship_instance.distributive_relationshi
p*)?
                                                                  >
<!ATTLIST EXPRESS_occurence.compound_relationship_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  simple_relationship_instance                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.simple_relationship_instance (XMI.extension*,

EXPRESS_occurence.attribute_instance.instance_of?,

EXPRESS_occurence.attribute_instance.its_value?,

EXPRESS_occurence.attribute_instance.in_partial_instance?)?
                                                               >
<!ATTLIST EXPRESS_occurence.simple_relationship_instance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  occurrence                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT EXPRESS_occurence.occurrence.base_type
(EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |
```

```
EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |

EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                    >

<!ELEMENT EXPRESS_occurence.occurrence.occurence_of
(EXPRESS_type_model.base_type |

EXPRESS_type_model.underlying_type |

EXPRESS_type_model.EXPRESS_type |

EXPRESS_type_model.constructed_type |

EXPRESS_type_model.select_type |

EXPRESS_type_model.enumeration_type |

EXPRESS_type_model.aggregation_type |

EXPRESS_type_model.bag_type |

EXPRESS_type_model.list_type |

EXPRESS_type_model.set_type |

EXPRESS_type_model.array_type |

EXPRESS_type_model.simple_type |

EXPRESS_type_model.logical_type |

EXPRESS_type_model.boolean_type |

EXPRESS_type_model.binary_type |

EXPRESS_type_model.number_type |

EXPRESS_type_model.real_type |

EXPRESS_type_model.integer_type |
```

```
EXPRESS_type_model.string_type |

EXPRESS_type_model.EXPRESS_entity)*
                                                    >

<!ELEMENT EXPRESS_occurence.occurrence (XMI.extension*,

EXPRESS_occurence.occurrence.base_type*,

EXPRESS_occurence.occurrence.occurence_of*)?
                                                 >
<!ATTLIST EXPRESS_occurence.occurrence
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT EXPRESS_occurence ((EXPRESS_occurence.instance |
                              EXPRESS_occurence.simple_value |
                              EXPRESS_occurence.aggregate_instance |
                              EXPRESS_occurence.element |
                              EXPRESS_occurence.partial_entity_instance |
                              EXPRESS_occurence.attribute_instance |
                              EXPRESS_occurence.relationship_instance |
                              EXPRESS_occurence.property_instance |

EXPRESS_occurence.compound_relationship_instance |

EXPRESS_occurence.simple_relationship_instance |
                              EXPRESS_occurence.occurrence)*) >
<!ATTLIST EXPRESS_occurence
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT expr_sem_n228 ((EXPRESS_schema | EXPRESS_type_model |
                          EXPRESS_constraints | EXPRESS_attribute_model |
                          EXPRESS_occurence)*) >
<!ATTLIST expr_sem_n228
            %XMI.element.att;
            %XMI.link.att;
>
```

Instance Document Using EXPRESS Semantic Model XMI Markup

For this example, the data is divided into two documents, one for the schema and one for data instances. This demonstrates the inherent ability in XMI to use multiple documents because it uses XPointers for addressing, which are inherently cross-document capable. It also reflects what would probably be typical practice, namely making schema instances into separate documents so that they are more easily re-used.

Schema document "pnp-schema.xml":

```
<?xml version="1.0"?>
<!DOCTYPE XMI SYSTEM "expr_sem_n228_xmi.dtd">
<XMI>
 <XMI.header>
  <XMI.documentation>
   <XMI.contact>W. Eliot Kimber</XMI.contact>
   <XMI.shortDescription>EXPRESS Schema for People and Pets Model. The
types defined in this document are referenced from instance documents.
   </XMI.shortDescription>
```

```
  </XMI.documentation>
  <XMI.model xmi.name="people_and_pets.exp"
xmi.version="0.3">expr_sem_n228_uml.mdl</XMI.model>
  <XMI.metamodel xmi.name="expr_sem_n228_uml.mdl">None
defined</XMI.metamodel>
 </XMI.header>
 <XMI.content>
  <expr_sem_n228>
   <EXPRESS_schema>
    <EXPRESS_schema.schema>

<EXPRESS_schema.schema.name>people_and_pets</EXPRESS_schema.schema.name>
     <EXPRESS_schema.schema.defined_types>
      <EXPRESS_type_model.EXPRESS_type xmi.id="type_name">
       <EXPRESS_type_model.EXPRESS_type.type_name>name
       </EXPRESS_type_model.EXPRESS_type.type_name>
       <EXPRESS_type_model.EXPRESS_type.based_on>
        <EXPRESS_type_model.string_type/>
       </EXPRESS_type_model.EXPRESS_type.based_on>
      </EXPRESS_type_model.EXPRESS_type>
      <EXPRESS_type_model.EXPRESS_type>
       <EXPRESS_type_model.named_types.select_type>
        <EXPRESS_type_model.select_type xmi.id="type_pet_animals">
         <EXPRESS_type_model.select_type.type_name>pet_animals
         </EXPRESS_type_model.select_type.type_name>
         <EXPRESS_type_model.select_type.selections>
          <EXPRESS_type_model.EXPRESS_entity href="#id(entity_dog)"/>
          <EXPRESS_type_model.EXPRESS_entity href="#id(entity_dog)"/>
         </EXPRESS_type_model.select_type.selections>
        </EXPRESS_type_model.select_type>
       </EXPRESS_type_model.named_types.select_type>
      </EXPRESS_type_model.EXPRESS_type>
      <EXPRESS_type_model.EXPRESS_entity xmi.id="entity_animal">
       <EXPRESS_type_model.EXPRESS_entity.entity_name>animal
       </EXPRESS_type_model.EXPRESS_entity.entity_name>
      </EXPRESS_type_model.EXPRESS_entity>
      <EXPRESS_type_model.EXPRESS_entity xmi.id="entity_human">
       <EXPRESS_type_model.EXPRESS_entity.entity_name>human
       </EXPRESS_type_model.EXPRESS_entity.entity_name>
      </EXPRESS_type_model.EXPRESS_entity>
      <EXPRESS_type_model.EXPRESS_entity xmi.id="entity_dog">
       <EXPRESS_type_model.EXPRESS_entity.entity_name>dog
       </EXPRESS_type_model.EXPRESS_entity.entity_name>
      </EXPRESS_type_model.EXPRESS_entity>
      <EXPRESS_type_model.EXPRESS_entity xmi.id="entity_cat">
       <EXPRESS_type_model.EXPRESS_entity.entity_name>cat
       </EXPRESS_type_model.EXPRESS_entity.entity_name>
      </EXPRESS_type_model.EXPRESS_entity>
      <EXPRESS_attribute_model.attribute>
       <EXPRESS_attribute_model.attribute.attribute_name>names
        </EXPRESS_attribute_model.attribute.attribute_name>
        <EXPRESS_attribute_model.attribute.defining_entity>
         <EXPRESS_type_model.EXPRESS_entity
href="#idref(entity_animal)"/>
        </EXPRESS_attribute_model.attribute.defining_entity>
        <EXPRESS_attribute_model.attribute.representation>
         <EXPRESS_type_model.EXPRESS_type href="#idref(type_name)"/>
        </EXPRESS_attribute_model.attribute.representation>
       </EXPRESS_attribute_model.attribute>
       <EXPRESS_attribute_model.attribute xmi.id="attribute_works_for">
        <EXPRESS_attribute_model.attribute.attribute_name>works_for
        </EXPRESS_attribute_model.attribute.attribute_name>
        <EXPRESS_attribute_model.attribute.defining_entity>
         <EXPRESS_type_model.EXPRESS_entity href="#id(entity_human)"/>
```

```
        </EXPRESS_attribute_model.attribute.defining_entity>
        <EXPRESS_attribute_model.attribute.representation>
         <EXPRESS_type_model.EXPRESS_entity href="#id(entity_human)"/>
         </EXPRESS_type_model.EXPRESS_entity>
        </EXPRESS_attribute_model.attribute.representation>
       </EXPRESS_attribute_model.attribute>
       <EXPRESS_attribute_model.attribute xmi.id="attribute_pets">
        <EXPRESS_attribute_model.attribute.attribute_name>pets
        </EXPRESS_attribute_model.attribute.attribute_name>
        <EXPRESS_attribute_model.attribute.defining_entity>
         <EXPRESS_type_model.EXPRESS_entity href="#id(entity_human)"/>
        </EXPRESS_attribute_model.attribute.defining_entity>
        <EXPRESS_attribute_model.attribute.representation>
         <EXPRESS_type_model.select_type href="#id(type_pet_animals)"/>
        </EXPRESS_attribute_model.attribute.representation>
       </EXPRESS_attribute_model.attribute>
       <EXPRESS_attribute_model.attribute
xmi.id="attribute_catches_frisbees">

<EXPRESS_attribute_model.attribute.attribute_name>catches_frisbees
        </EXPRESS_attribute_model.attribute.attribute_name>
        <EXPRESS_attribute_model.attribute.defining_entity>
         <EXPRESS_type_model.EXPRESS_entity href="#id(entity_dog)"/>
        </EXPRESS_attribute_model.attribute.defining_entity>
        <EXPRESS_attribute_model.attribute.representation>
         <EXPRESS_type_model.boolean_type/>
        </EXPRESS_attribute_model.attribute.representation>
       </EXPRESS_attribute_model.attribute>
       <EXPRESS_attribute_model.attribute
xmi.id="attribute_addicted_to_catnip">

<EXPRESS_attribute_model.attribute.attribute_name>addicted_to_catnip
        </EXPRESS_attribute_model.attribute.attribute_name>
        <EXPRESS_attribute_model.attribute.defining_entity>
         <EXPRESS_type_model.EXPRESS_entity href="#id(entity_cat)"/>
        </EXPRESS_attribute_model.attribute.defining_entity>
        <EXPRESS_attribute_model.attribute.representation>
         <EXPRESS_type_model.boolean_type/>
        </EXPRESS_attribute_model.attribute.representation>
       </EXPRESS_attribute_model.attribute>
      </EXPRESS_schema.schema.defined_types>
     </EXPRESS_schema.schema>
    </EXPRESS_schema>
   </expr_sem_n228>
  </XMI.content>
</XMI>
```

Entity instance population document "pnp-sample2.xml":

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xmi SYSTEM "expr_sem_n228_xmi.dtd" [
]>
<xmi>
<xmi.header>
<xmi.documentation>
<xmi.contact>W. Eliot Kimber</xmi.contact>
<xmi.shortdescription>Late Bound Instance of People and Pets
Model</xmi.shortdescription>
</xmi.documentation>
<xmi.model xmi.name="people_and_pets.exp"
xmi.version="0.3">people_and_pets.exp
</xmi.model>
<xmi.metamodel xmi.name="expr_sem_n228_uml.mdl">expr_sem_n228_uml.mdl
</xmi.metamodel>
```

```
</xmi.header>
<xmi.content>
<expr_sem_n228>
<express_occurence>
<express_occurence.instance xmi.id="oid001">
<express_occurence.instance.oid>oid001</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_names)"/>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>W. Eliot Kimber
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity href="pnp-
schema.xml#id(entity_animal)"/>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.relationship_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_works_for)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.instance
href="#id(oid002)">
</express_occurence.instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.relationship_instance>
<express_occurence.relationship_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_pets)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.instance
href="#id(oid003)">
</express_occurence.instance>
</express_occurence.element.element_value>
```

```
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.instance href="#id(oid004)">
</express_occurence.instance>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.instance href="#id(oid005)">
</express_occurence.instance>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.relationship_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity href="pnp-
schema.xml#id(entity_human)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance xmi.id="oid002">
<express_occurence.instance.oid>oid002
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Renee Swank
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.relationship_instance>
<express_occurence.attribute_instance.instance_of>
```

```
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_works_for)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.instance
href="#id(oid006)">
</express_occurence.instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.relationship_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_human)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid003">
<express_occurence.instance.oid>oid003
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Sigfried
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_addicted_to_catnip)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.simple_value>
```

```
<express_occurence.simple_value.data>True
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_cat)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid004">
<express_occurence.instance.oid>oid004
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Bete Noir
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Beteski
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_addicted_to_catnip)">
</express_attribute_model.explicit_attribute>
```

```
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>True
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_cat)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid005">
<express_occurence.instance.oid>oid005
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Forrest
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Dogboy
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Noseboy
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
```

```
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_catches_frisbees)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>False
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_dog)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid006">
<express_occurence.instance.oid>oid006
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Carla Corkern
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Task
Mistress
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
```

```
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.relationship_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_works_for)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.instance
href="#id(oid007)">
</express_occurence.instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.relationship_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_human)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid007">
<express_occurence.instance.oid>oid007
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Lucie Feldstat
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
```

```
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.relationship_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_works_for)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.instance
href="#id(oid007)">
</express_occurence.instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.relationship_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_human)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid008">
<express_occurence.instance.oid>oid008
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Chauncy
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>The Chaunce
Man
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
```

```
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_catches_frisbees)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>False
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_dog)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
<express_occurence.instance
xmi.id="oid009">
<express_occurence.instance.oid>oid009
</express_occurence.instance.oid>
<express_occurence.instance.components>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute
href="pnp-schema.xml#id(attribute_names)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.aggregate_instance>
<express_occurence.aggregate_instance.elements>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>Chelsie
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
<express_occurence.element>
<express_occurence.element.element_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>The Chelster
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.element.element_value>
</express_occurence.element>
</express_occurence.aggregate_instance.elements>
</express_occurence.aggregate_instance>
</express_occurence.attribute_instance.its_value>
```

```
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_animal)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance>
<express_occurence.partial_entity_instance.attribute_values>
<express_occurence.property_instance>
<express_occurence.attribute_instance.instance_of>
<express_attribute_model.explicit_attribute href="pnp-
schema.xml#id(attribute_catches_frisbees)">
</express_attribute_model.explicit_attribute>
</express_occurence.attribute_instance.instance_of>
<express_occurence.attribute_instance.its_value>
<express_occurence.simple_value>
<express_occurence.simple_value.data>False
</express_occurence.simple_value.data>
</express_occurence.simple_value>
</express_occurence.attribute_instance.its_value>
</express_occurence.property_instance>
</express_occurence.partial_entity_instance.attribute_values>
<express_occurence.partial_entity_instance.instance_of>
<express_type_model.express_entity
href="pnp-schema.xml#id(entity_dog)">
</express_type_model.express_entity>
</express_occurence.partial_entity_instance.instance_of>
</express_occurence.partial_entity_instance>
</express_occurence.instance.components>
</express_occurence.instance>
</express_occurence>
</expr_sem_n228>
</xmi.content>
</xmi>
```

4.1.4. XMI DTD for EXPRESS SDAI Model

This section presents the XMI DTD generated from the UML transliteration of the EXPRESS SDAI model. The common XMI-defined declarations have been omitted for brevity. This DTD is presented here for the sake of completeness. There is no corresponding instance data as the project scope did not allow time for its creation.

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL:  data_express_by_aggregation_01_wek                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT schema_definition.entity_definition
                                                (types.entity_definition)*
>

<!ELEMENT schema_definition.defined_type (types.defined_type)* >

<!ELEMENT schema_definition.global_rule (global_rule)* >

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  schema_definition                                  -->
```

```
<!-- ____________________________________________________________________ -->


<!ELEMENT schema_definition.name (types.express_id) >

<!ELEMENT schema_definition.constant_definition (constant_definition)* >

<!ELEMENT schema_definition.named_type (types.named_type |
                                        types.defined_type |
                                        types.entity_definition)* >

<!ELEMENT schema_definition (schema_definition.name?, XMI.extension*,
                             schema_definition.constant_definition*,
                             schema_definition.global_rule*,
                             schema_definition.named_type*,
                             schema_definition.entity_definition*,
                             schema_definition.defined_type*,
                             schema_definition.global_rule*)? >
<!ATTLIST schema_definition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  constant_definition                                -->
<!-- ____________________________________________________________________ -->


<!ELEMENT constant_definition.constant_type (types.base_type |
                                             types.named_type |
                                             types.defined_type |
                                             types.entity_definition |
                                             types.aggregation_type |
                                             types.set_type |
                                             types.list_type |
                                             types.array_type |
                                             types.bag_type |
                                             types.constructed_type |
                                             types.enumeration_type |
                                             types.select_type |
                                             types.simple_type |
                                             types.integer_type |
                                             types.boolean_type |
                                             types.binary_type |
                                             types.string_type |
                                             types.number_type |
                                             types.logical_type |
                                             types.real_type) >

<!ELEMENT constant_definition.constant_expression (expression |
                                                   integer_expression |
                                                   logical_expression) >

<!ELEMENT constant_definition.name (types.express_id) >

<!ELEMENT constant_definition.schema_definition (schema_definition)? >

<!ELEMENT constant_definition.global_rule (global_rule)? >

<!ELEMENT constant_definition.function_definition
                                                   (function_definition)?
>
```

```
<!ELEMENT constant_definition.procedure_definition

(procedure_definition)? >

<!ELEMENT constant_definition (constant_definition.constant_type?,
                               constant_definition.constant_expression?,
                               constant_definition.name?,
                               XMI.extension*,
                               constant_definition.schema_definition?,
                               constant_definition.global_rule?,
                               constant_definition.function_definition?,

constant_definition.procedure_definition?)?
                                >
<!ATTLIST constant_definition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  global_rule                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT global_rule.name (types.express_id) >

<!ELEMENT global_rule.parent_schema (schema_definition) >

<!ELEMENT global_rule.schema_definition (schema_definition)? >

<!ELEMENT global_rule.where_rule (where_rule)* >

<!ELEMENT global_rule.entity_definition (types.entity_definition)* >

<!ELEMENT global_rule.constant_definition (constant_definition)* >

<!ELEMENT global_rule (global_rule.name?, global_rule.parent_schema?,
                       XMI.extension*, global_rule.schema_definition?,
                       global_rule.where_rule*,
                       global_rule.entity_definition*,
                       global_rule.constant_definition*)? >
<!ATTLIST global_rule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  where_rule                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT where_rule.name (types.express_id) >

<!ELEMENT where_rule.the_expression (logical_expression) >

<!ELEMENT where_rule.named_type (types.named_type | types.defined_type |
                                 types.entity_definition)? >

<!ELEMENT where_rule.global_rule (global_rule)? >

<!ELEMENT where_rule (where_rule.name?, where_rule.the_expression?,
                      XMI.extension*, where_rule.named_type?,
```

```
                        where_rule.global_rule?)? >
<!ATTLIST where_rule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  interface_specification                            -->
<!-- ____________________________________________________________________ -->


<!ELEMENT interface_specification (XMI.extension*)? >
<!ATTLIST interface_specification
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  expression                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT expression (XMI.extension*)? >
<!ATTLIST expression
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  function_definition                                -->
<!-- ____________________________________________________________________ -->


<!ELEMENT function_definition.name (types.express_id) >

<!ELEMENT function_definition.parent_schema (schema_definition) >

<!ELEMENT function_definition.constant_definition
                                                    (constant_definition)*
>

<!ELEMENT function_definition (function_definition.name?,
                               function_definition.parent_schema?,
                               XMI.extension*,
                               function_definition.constant_definition*)?
                                >
<!ATTLIST function_definition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  procedure_definition                               -->
<!-- ____________________________________________________________________ -->


<!ELEMENT procedure_definition.name (types.express_id) >

<!ELEMENT procedure_definition.parent_schema (schema_definition) >
```

```
<!ELEMENT procedure_definition.constant_definition

(constant_definition)* >

<!ELEMENT procedure_definition (procedure_definition.name?,
                                procedure_definition.parent_schema?,
                                XMI.extension*,

procedure_definition.constant_definition*)?
                                 >
<!ATTLIST procedure_definition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  uniqueness_rule                                    -->
<!-- ____________________________________________________________________ -->


<!ELEMENT uniqueness_rule.name (types.express_id) >

<!ELEMENT uniqueness_rule.entity_definition (types.entity_definition)? >

<!ELEMENT uniqueness_rule.attribute (attribute | inverse_attribute |
                                     derived_attribute |
                                     explicit_attribute)* >

<!ELEMENT uniqueness_rule (uniqueness_rule.name?, XMI.extension*,
                           uniqueness_rule.entity_definition?,
                           uniqueness_rule.attribute*)? >
<!ATTLIST uniqueness_rule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  integer_expression                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT integer_expression (XMI.extension*)? >
<!ATTLIST integer_expression
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  logical_expression                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT logical_expression (XMI.extension*)? >
<!ATTLIST logical_expression
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  attribute                                          -->
```

```
<!-- ____________________________________________________________________ -->


<!ELEMENT attribute.name (types.express_id) >

<!ELEMENT attribute.entity_definition (types.entity_definition)? >

<!ELEMENT attribute.uniqueness_rule (uniqueness_rule)? >

<!ELEMENT attribute (attribute.name?, XMI.extension*,
                     attribute.entity_definition?,
                     attribute.uniqueness_rule?)? >
<!ATTLIST attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  inverse_attribute                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT inverse_attribute.inverted_attribute (attribute |
                                                inverse_attribute |
                                                derived_attribute |
                                                explicit_attribute) >

<!ELEMENT inverse_attribute.entity_definition
                                               (types.entity_definition)?
>

<!ELEMENT inverse_attribute (attribute.name?,
                             inverse_attribute.inverted_attribute?,
                             XMI.extension*,
                             attribute.entity_definition?,
                             attribute.uniqueness_rule?,
                             inverse_attribute.entity_definition?)? >
<!ATTLIST inverse_attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  derived_attribute                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT derived_attribute.domain (types.base_type | types.named_type |
                                    types.defined_type |
                                    types.entity_definition |
                                    types.aggregation_type |
                                    types.set_type | types.list_type |
                                    types.array_type | types.bag_type |
                                    types.constructed_type |
                                    types.enumeration_type |
                                    types.select_type |
                                    types.simple_type |
                                    types.integer_type |
                                    types.boolean_type |
                                    types.binary_type |
                                    types.string_type |
                                    types.number_type |
```

```
                                    types.logical_type |
                                    types.real_type) >

<!ELEMENT derived_attribute (attribute.name?,
                             derived_attribute.domain?, XMI.extension*,
                             attribute.entity_definition?,
                             attribute.uniqueness_rule?)? >
<!ATTLIST derived_attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  explicit_attribute                                 -->
<!-- ____________________________________________________________________ -->


<!ELEMENT explicit_attribute.domain (types.base_type | types.named_type |
                                     types.defined_type |
                                     types.entity_definition |
                                     types.aggregation_type |
                                     types.set_type | types.list_type |
                                     types.array_type | types.bag_type |
                                     types.constructed_type |
                                     types.enumeration_type |
                                     types.select_type |
                                     types.simple_type |
                                     types.integer_type |
                                     types.boolean_type |
                                     types.binary_type |
                                     types.string_type |
                                     types.number_type |
                                     types.logical_type |
                                     types.real_type) >

<!ELEMENT explicit_attribute (attribute.name?,
                              explicit_attribute.domain?,
                              XMI.extension*,
                              attribute.entity_definition?,
                              attribute.uniqueness_rule?)? >
<!ATTLIST explicit_attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL PACKAGE:  types                                            -->
<!-- ____________________________________________________________________ -->


<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  base_type                                          -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.base_type (XMI.extension*)? >
<!ATTLIST types.base_type
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  named_type                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.named_type.name (types.express_id) >

<!ELEMENT types.named_type.parent_schema (schema_definition) >

<!ELEMENT types.named_type.where_rule (where_rule)* >

<!ELEMENT types.named_type.select_type (types.select_type)* >

<!ELEMENT types.named_type.schema_definition (schema_definition)? >

<!ELEMENT types.named_type (types.named_type.name?,
                            types.named_type.parent_schema?,
                            XMI.extension*,
                            types.named_type.where_rule*,
                            types.named_type.select_type*,
                            types.named_type.schema_definition?)? >
<!ATTLIST types.named_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  logical_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.logical_type (XMI.extension*)? >
<!ATTLIST types.logical_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  boolean_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.boolean_type (XMI.extension*)? >
<!ATTLIST types.boolean_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  defined_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.defined_type.domain (types.underlying_type |
                                     types.simple_type |
                                     types.integer_type |
                                     types.boolean_type |
                                     types.binary_type |
                                     types.string_type |
                                     types.number_type |
```

```
                                        types.logical_type |
                                        types.real_type |
                                        types.defined_type |
                                        types.constructed_type |
                                        types.enumeration_type |
                                        types.select_type |
                                        types.named_type |
                                        types.entity_definition |
                                        types.aggregation_type |
                                        types.set_type | types.list_type |
                                        types.array_type | types.bag_type) >

<!ELEMENT types.defined_type.schema_definition (schema_definition)? >

<!ELEMENT types.defined_type (types.named_type.name?,
                              types.named_type.parent_schema?,
                              types.defined_type.domain?,
                              XMI.extension*,
                              types.named_type.where_rule*,
                              types.named_type.select_type*,
                              types.named_type.schema_definition?,
                              types.defined_type.schema_definition?)? >
<!ATTLIST types.defined_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  aggregation_type                                   -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.aggregation_type.element_type (types.base_type |
                                               types.named_type |
                                               types.defined_type |
                                               types.entity_definition |
                                               types.aggregation_type |
                                               types.set_type |
                                               types.list_type |
                                               types.array_type |
                                               types.bag_type |
                                               types.constructed_type |
                                               types.enumeration_type |
                                               types.select_type |
                                               types.simple_type |
                                               types.integer_type |
                                               types.boolean_type |
                                               types.binary_type |
                                               types.string_type |
                                               types.number_type |
                                               types.logical_type |
                                               types.real_type) >

<!ELEMENT types.aggregation_type (types.aggregation_type.element_type?,
                                  XMI.extension*)? >
<!ATTLIST types.aggregation_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  select_type                                        -->
```

```
<!-- ____________________________________________________________________ -->


<!ELEMENT types.select_type.named_type (types.named_type |
                                        types.defined_type |
                                        types.entity_definition)* >

<!ELEMENT types.select_type (XMI.extension*,
                             types.select_type.named_type*)? >
<!ATTLIST types.select_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  set_type                                           -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.set_type (types.aggregation_type.element_type?,
                          XMI.extension*)? >
<!ATTLIST types.set_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  list_type                                          -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.list_type (types.aggregation_type.element_type?,
                           XMI.extension*)? >
<!ATTLIST types.list_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  array_type                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.array_type (types.aggregation_type.element_type?,
                            XMI.extension*)? >
<!ATTLIST types.array_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  bag_type                                           -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.bag_type (types.aggregation_type.element_type?,
                          XMI.extension*)? >
<!ATTLIST types.bag_type
            %XMI.element.att;
            %XMI.link.att;
```

```
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  enumeration_type                                   -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.enumeration_type.express_id (types.express_id)* >

<!ELEMENT types.enumeration_type (XMI.extension*,
                                  types.enumeration_type.express_id*)? >
<!ATTLIST types.enumeration_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  express_id                                         -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.express_id.enumeration_type (types.enumeration_type)? >

<!ELEMENT types.express_id (XMI.extension*,
                            types.express_id.enumeration_type?)? >
<!ATTLIST types.express_id
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  constructed_type                                   -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.constructed_type (XMI.extension*)? >
<!ATTLIST types.constructed_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  number_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.number_type (XMI.extension*)? >
<!ATTLIST types.number_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  underlying_type                                    -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.underlying_type (XMI.extension*)? >
<!ATTLIST types.underlying_type
```

```
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  integer_type                                       -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.integer_type (XMI.extension*)? >
<!ATTLIST types.integer_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  binary_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.binary_type (XMI.extension*)? >
<!ATTLIST types.binary_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  string_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.string_type (XMI.extension*)? >
<!ATTLIST types.string_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  real_type                                          -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.real_type (XMI.extension*)? >
<!ATTLIST types.real_type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  simple_type                                        -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.simple_type (XMI.extension*)? >
<!ATTLIST types.simple_type
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  entity_definition                                  -->
<!-- ____________________________________________________________________ -->


<!ELEMENT types.entity_definition.attribute (attribute |
                                             inverse_attribute |
                                             derived_attribute |
                                             explicit_attribute)* >

<!ELEMENT types.entity_definition.global_rule (global_rule)* >

<!ELEMENT types.entity_definition.entity_definition

(types.entity_definition)* >

<!ELEMENT types.entity_definition.uniqueness_rule (uniqueness_rule)* >

<!ELEMENT types.entity_definition.inverse_attribute
                                                    (inverse_attribute)*
>

<!ELEMENT types.entity_definition.schema_definition
                                                    (schema_definition)?
>

<!ELEMENT types.entity_definition (types.named_type.name?,
                                   types.named_type.parent_schema?,
                                   XMI.extension*,
                                   types.named_type.where_rule*,
                                   types.named_type.select_type*,
                                   types.named_type.schema_definition?,
                                   types.entity_definition.attribute*,
                                   types.entity_definition.global_rule*,

types.entity_definition.entity_definition*,

types.entity_definition.uniqueness_rule*,

types.entity_definition.inverse_attribute*,

types.entity_definition.schema_definition?)?
                                    >
<!ATTLIST types.entity_definition
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT types ((types.base_type | types.named_type |
                  types.logical_type | types.boolean_type |
                  types.defined_type | types.aggregation_type |
                  types.select_type | types.set_type | types.list_type |
                  types.array_type | types.bag_type |
                  types.enumeration_type | types.express_id |
                  types.constructed_type | types.number_type |
                  types.underlying_type | types.integer_type |
                  types.binary_type | types.string_type | types.real_type
|
                  types.simple_type | types.entity_definition)*) >
<!ATTLIST types
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!ELEMENT data_express_by_aggregation_01_wek ((schema_definition |
                                               constant_definition |
                                               global_rule | where_rule |
                                               interface_specification |
                                               expression |
                                               function_definition |
                                               procedure_definition |
                                               uniqueness_rule |
                                               integer_expression |
                                               logical_expression |
                                               attribute |
                                               inverse_attribute |
                                               derived_attribute |
                                               explicit_attribute |
                                               types)*) >
<!ATTLIST data_express_by_aggregation_01_wek
            %XMI.element.att;
            %XMI.link.att;
>
```

4.1.5. Critique of the XMI Design

The XMI design appears to meet its stated requirements as being a mechanism for the XML-based interchange of MOF-based models and instances. However, it does not work as well as a representation mechanism for EXPRESS-based data for the simple reason that there is not a clean mapping between EXPRESS and UML. XMI lacks some generalizations or representation options that would make it better suited to the representation EXPRESS data instances. In addition, the XMI design relies entirely on element type names and construction algorithms to map elements to the MOF or model constructs they represent. This is a fairly weak design approach and tends to limit the flexibility and extensibility of the representation syntax. The use of XLinks in the XMI design is somewhat problematical in its details, although it is fundamentally a sound design approach.

XMI clearly needs a generic mechanism for defining data types for attribute values. Because XMI only recognizes the CORBA-defined data types, it provides no satisfying or direct method for reflecting data type names from the original model in the XMI form. I did not find XMI’s justification for not having a more generic data typing mechanism to be compelling. The design decision as given in the XI specification is:

"Design decision: The XMI proposal does not map all MOF DataTypes onto distinct elements in the XMI DTD.

"Encoding of MOF DataTypes (i.e., CORBA data types) in XMI presents us with a tricking problem. It is arguable that XMI should map data types so that the XMI DTDs allow full validation. However, if XMI were to do this, there is a substantial risk that future integrate[sic] with the XML proposals being developed by W3C would be problematical. XMI therefore optionally encodes most CORBA data types using "boilerplate" DTDs. It is anticipated that this decision will be revisited in the future."

The problem with this statement is twofold. First, the primary purpose of representing data types is labeling, not validation. I presume that by "full validation" the XMI designers mean validation of the data value to the data type. This will probably never be possible in an XML-only context as it is unlikely that any future XML schema mechanism will provide a sufficiently complete data typing and validation mechanism to meet this requirement generally (for example, it would need to encompass the same level of descriptive power as EXPRESS for the definition of constraints on data values). However, just by providing a way to create element type names that reflect data types in the source models, XMI would enable the ability to map attribute values to validation processes outside of an XML parsing context. The current XMI design does not allow even this level of potential validation.

The XMI design uses element type names and containment exclusively to map elements to their corresponding MOF or model-specific components. For example, given the element type name "foo.bar.baz", it is impossible to know if this element is an instance of the class "baz" in the package "bar" in the package "foo" or the attribute "baz" of the class "bar" in the package "foo" or the association role "baz" for the class "bar" in the package "foo" unless you know its precise context within the document and have access to the original model. Likewise, within a class element, it is impossible to tell by looking where the properties start and the associations and containments end. As a rule, it is much better to explicitly label each element with the type of thing it corresponds to, e.g., "object" (instance of a class), "attribute", "simple type", "association role", etc. This makes the syntax-to-abstraction mapping obvious and enables generic processing. It also frees the markup design to provide a variety of structural choices without losing the ability to process the data correctly. It would be a very simple addition to the XMI design to add this. The base types to use for the labels are already defined in the MOF.

It is also useful to use attributes to capture component names in addition to using element type names. This is because element type names must often be unnecessarily long or complex, as is the case in XMI, simply to satisfy XML's uniqueness rules. For example, given the attribute element "foo.bar.baz", there is no way to get back the original attribute name except by parsing the element type name. It would be much simpler and clearer if the element had an attribute such as `"attname="baz"`. In practical terms, it is much easier for most XML processing and retrieval tools to resolve the query "select * from elements where type='attribute' and name='baz'" than the query "select * from element where tagName matches '\.baz$'" for the simple reason that most XML tools are optimized for selecting elements by whole element type name or by attribute value and either do not provide direct ways to do pattern matching of element type names or impose serious processing overhead. In any case, the people building the queries must know the syntax rules for the element type names, which cannot be defined using any XML-defined method.

Putting component names in attributes also relaxes constraints on element type names, allowing, for example, different element type names for the same abstract thing or multiple synonymous names for a single object.

The XMI design uses element containment to reflect all associations. This is a perfectly reasonable design but it does require the use of some mechanism to clearly distinguish references to things from the things themselves. The XMI design overloads class elements with being either things or pointers to things. One problem is that while the XMI specification is clear on the meaning of these links, nothing in the current XLink specification provides a clear syntactic mechanism for expressing the desired use-by-reference semantic. Nor does XML provide a way to enforce the "no content if you're a hyperlink" rule (SGML's "CONREF" feature does allow this, but XML did not include this feature for the simple reason that it prevents the ability to parse instances without having and processing the attribute list declarations). The desired semantics can be defined using the value reference facility of the HyTime standard, ISO/IEC 10744:1997, while still allowing the use of XPointers and XLink.

In doing XML design generally, I prefer to have distinct element types for things and references to things, as it tends to make the markup clearer to users. As the XMI specification appears to preclude the use of multiple non-proxy elements for the same abstract object instance, an XMI writer will always have to know when to create a reference and when to create the thing. Thus it must know the class the element represents and thus there would be no difficulty in implementing a markup schema that uses two different element type names, one for the thing and one for the reference for the same class. When using attributes to convey the actual class, there is no problem with having different element types represent the same instance of the a given class. This lets you have specific attribute sets and content models for each type of element, making it possible to define in the XMI DTD rules the constraints that can only be defined in prose in the current design.

The XMI design uses XLink for turning class elements into hyperlinks but is inconsistent in its use of both direct ID references and XPointers. The problem is that XLink only supports the use of XPointers from "href" attributes (that is, as part of URLs). This is because without an explicit DTD, it is impossible to know that a particular attribute is in fact an ID reference or that a particular attribute is in fact an ID. Thus, XLink has no choice but to limit itself to the use of XPointers. XMI should do the same. One potential consequence of the current design is that generic XLink processors will not give the desired result for links that use direct ID references. XMI should use XLinks and XPointers exclusively for all references.

The provision for universally unique identifier (UUID) attributes on class elements would appear to be redundant with the normal XMI ID mechanism. However, it appears to make sense if the intent is that two class elements with the same UUID are in fact copies of the same object instance. It also makes sense if the intent is to provide a way for repository-specific object identifiers to survive through an XMI interchange transaction. However, in this latter case, XMI should provide a generic "object identifier" attribute that is not tied to a particular identifier syntax or scheme as the UUID attribute is.

The use of "ANY" content models is a bit heavy handed and somewhat unnecessary. However, it does make extension easier by essentially allowing anything anywhere in

many contexts. However, it is would probably be better to have more constrained content models with conventional parameter entities used to enable the easy addition of document-specific element types where they are appropriate. As a matter of XML design, ANY is almost never necessary or appropriate when the intent is otherwise to have a constraining document type, as appears to be the case for XMI. The use of SGML architectures would also make it possible to have unbounded extensions in local documents without the need to compromise the XMI-defined content rules.

4.2. Product Data Markup Language (PDIT)

The PDML Early Binding representation was designed initially to enable the interchange of EXPRESS entity instances in a heterogeneous, Web-based interchange environment, one in which people are likely to view the instance data directly in XML form through XML-aware Web browsers. Another design goal was to use XML DTDs to reflect the schema-defined constraints as much as possible, in large part to enable the validation of some or all of the schema constraints through normal XML validation. It was also a requirement to be complete with respect to EXPRESS such that any valid EXPRESS schema could be represented using the PDML markup approach. The original design of PDML was limited to single-schema models, although it was concluded at the end of the second Early Binding Workshop that multi-schema models could be handled by the existing design given a way to bind element types to specific schemas.

PDIT, Inc. has implemented a PDML DTD generator and made it publicly available. The PDML project is funded by the U.S. DoD.

The PDML Early Binding design has been refined through a series of two workshops sponsored by the DoD JECPO in order to prepare it as a technology submission for consideration for the Part 28 effort. These workshops have done much to improve the general understanding of EXPRESS representation issues and the use of XML. As several members of the Part 28 Early Binding working group were involved in these workshops, the Early Binding effort has had a direct and positive affect on the Late Binding design.

4.2.1. Early Binding Mapping Rules

The PDML early binding mapping rules are similar to the XMI rules, but specific to the EXPRESS language.

In summary, the mapping rules are:

- Each entity type is represented by an element of the same name. If there are multiple schemas in the model, the schema name is prepended to the type name, separated by a period (".")
- The content of each entity element consists of any subtype elements, followed by the attributes for the type.

- Each attribute is represented by an element whose element type name is the entity type name prepended to the attribute name, separated by a period.
- Attribute values are contained by elements whose element type name is the same as the data type name (this is possible because the entity and data types form a single type name space).
- Attribute values that are aggregate types are represented by elements whose names reflect the type of aggregation (list, set, bag, or array), the cardinality, and the data type of the aggregate members.
- Select types are represented by element types whose name is the name of the select type and whose content is the types allowed.
- For each entity type, there is a corresponding "-ref" element type used to represent references to entities of that type (contrast with the XMI approach of using the same element type for instances and references to instances)

4.2.2. PDML Early Bound DTD For People and Pets Model

These declarations show the DTD that results from applying the PDML mapping rules to the people and pets EXPRESS model. These declarations were created by hand, not by a tool.

```
<!-- DTD declarations for early-bound documents for model using
     people_and_pets schema -->
<!-- Created by hand, not by EXML -->

<?IS10744 arch name="ebarch"
   public-id="Early binding architecture"
   dtd-system-id="early-binding-arch.dtd"
   doc-elem-form="model-instance-set"
?>

<?IS10744 ArcBase ebarch ?>
<!NOTATION ebarch PUBLIC "Early binding architecture" >
<!ATTLIST #NOTATION ebarch
   ArcDTD
     CDATA
     #FIXED "%ebarch-dtd"
   ArcDocF
     CDATA
     #FIXED "model-instance-set"
>

<!NOTATION express  -- EXPRESS language schema --
  PUBLIC "ISO 10303-11:1992::NOTATION EXPRESS Language Reference//EN"
>

<!ENTITY % ebarch-dtd SYSTEM "early-binding-arch.dtd" >

<!ENTITY % built-in-types SYSTEM "built-in-types.dtd" >
%built-in-types;

<!ENTITY % defined-top-level-entity-types
  "animal"
```

```
>


<!ELEMENT people_and_pets
  (constant-instances,
   non-constant-instances)
>
<!ATTLIST people_and_pets
   id
     ID
     #IMPLIED
   schema-names -- Blank-delimited list of schema names --
     CDATA
     #REQUIRED
   hub-schema   -- Name of root schema for model --
     CDATA
     #REQUIRED
   express-schemas  -- Pointer to actual EXPRESS schema file --
     ENTITIES       -- External data entity with a notation of "express"
--
     #IMPLIED       -- Specify only if schema exists in EXPRESS form --
   ebarch
     CDATA
     #FIXED "model-instance-set"
>

<!ELEMENT (constant-instances | non-constant-instances)
  (%defined-top-level-entity-types;)*
>
<!ATTLIST constant-instances
  ebarch
    CDATA
    #FIXED "constant-instances"
>
<!ATTLIST non-constant-instances
  ebarch
    CDATA
    #FIXED "non-constant-instances"
>

<!ELEMENT name
  (string)
>
<!ATTLIST name
  ebarch
     CDATA
     #FIXED "defined-datatype-instance"
  express-source-name
     CDATA
     #FIXED "name"
>
<!ELEMENT pet_animals
  (cat-ref |
   dog-ref)
>
<!ATTLIST pet_animals
  ebarch
     CDATA
     #FIXED "select"
  express-source-name
     CDATA
     #FIXED "pet_animals"
>
```

```
<!ELEMENT animal
  (animal.names,
   animal.subtypes)
>
<!ATTLIST animal
  ebarch
     CDATA
     #FIXED "nested-complex-entity-instance"
  express-source-name
     CDATA
     #FIXED "animal"
  governing-schema
     CDATA
     #FIXED "people_and_pets"
>
<!ELEMENT animal.names
  (list-1-n-of-name)
>
<!ATTLIST animal.names
  ebarch
     CDATA
     #FIXED "attribute"
  express-source-name
     CDATA
     #FIXED "names"
>

<!ELEMENT animal.subtypes
  (human |
   cat |
   dog)
>
<!ATTLIST animal-subtypes
  ebarch
     CDATA
     #FIXED "subtypes"
>

<!ELEMENT human
  (human.pets?,
   human.works_for)
>
<!ATTLIST human
  id
     ID
     #IMPLIED
  ebarch
     CDATA
     #FIXED "nested-partial-entity-instance"
  express-source-name
     CDATA
     #FIXED "human"
  governing-schema
     CDATA
     #FIXED "people_and_pets"
>

<!ELEMENT human-ref
  EMPTY
>
<!ATTLIST human-ref
  entity-instance
    IDREF
    #REQUIRED
```

```
  id
     ID
     #IMPLIED
  ebarch
     CDATA
     #FIXED "entity-instance-ref"
  loctype
     CDATA
     #FIXED "entity-instance IDLOC"
>

<!ELEMENT human.works_for
  (human-ref)
>
<!ATTLIST human.works_for
  ebarch
     CDATA
     #FIXED "attribute"
  express-source-name
     CDATA
     #FIXED "works_for"
>

<!ELEMENT human.pets
  (list-1-n-of-pet_animals)
>
<!ATTLIST human.pets
  ebarch
     CDATA
     #FIXED "attribute"
  express-source-name
     CDATA
     #FIXED "pets"
>

<!ELEMENT list-1-n-of-pet_animals
  (pet_animals+)
>
<!ATTLIST list-1-n-of-pet_animals
  ebarch
     CDATA
     #FIXED "list"
  express-source-name
     CDATA
     #FIXED "LIST [1:?] OF pet_animals"
>

<!ELEMENT list-1-n-of-name
  (name+)
>
<!ATTLIST list-1-n-of-name
  ebarch
     CDATA
     #FIXED "list"
  express-source-name
     CDATA
     #FIXED "LIST [1:?] OF name"
>

<!ELEMENT dog
  (dog.catches_frisbees)
>
<!ATTLIST dog
  id
```

```
      ID
      #IMPLIED
  ebarch
      CDATA
      #FIXED "nested-partial-entity-instance"
  express-source-name
      CDATA
      #FIXED "dog"
  governing-schema
      CDATA
      #FIXED "people_and_pets"
>

<!ELEMENT dog-ref
  EMPTY
>
<!ATTLIST dog-ref
  entity-instance
    IDREF
    #REQUIRED
  id
      ID
      #IMPLIED
  ebarch
      CDATA
      #FIXED "entity-instance-ref"
  loctype
      CDATA
      #FIXED "entity-instance IDLOC"
>

<!ELEMENT dog.catches_frisbees
  (boolean)
>
<!ATTLIST dog.catches_frisbees
  ebarch
      CDATA
      #FIXED "attribute"
  express-source-name
      CDATA
      #FIXED "catches_frisbees"
>


<!ELEMENT cat
  (cat.addicted_to_catnip)
>
<!ATTLIST cat
  id
      ID
      #IMPLIED
  ebarch
      CDATA
      #FIXED "nested-partial-entity-instance"
  express-source-name
      CDATA
      #FIXED "cat"
  governing-schema
      CDATA
      #FIXED "people_and_pets"
>

<!ELEMENT cat-ref
  EMPTY
```

```
>
<!ATTLIST cat-ref
  entity-instance
    IDREF
    #REQUIRED
  id
     ID
     #IMPLIED
  ebarch
     CDATA
     #FIXED "entity-instance-ref"
  loctype
     CDATA
     #FIXED "entity-instance IDLOC"
>

<!ELEMENT cat.addicted_to_catnip
  (boolean)
>
<!ATTLIST cat.addicted_to_catnip
  ebarch
     CDATA
     #FIXED "attribute"
  express-source-name
     CDATA
     #FIXED "addicted_to_catnip"
>
```

4.2.3. PDML Early Bound Data Instance

```
<?xml version="1.0"?>
<!DOCTYPE people_and_pets SYSTEM "people_and_pets.dtd" [
<!ENTITY people_and_pets_schema SYSTEM "people-and-pets.exp" NDATA
express>
]>
<people_and_pets schema-names="people_and_pets" hub-
schema="people_and_pets" express-schemas="people_and_pets_schema">
<constant-instances></constant-instances>
<non-constant-instances>
 <animal>
  <animal.names>
   <list-1-n-of-name>
    <name>
     <string>W. Eliot Kimber</string>
    </name>
   </list-1-n-of-name>
  </animal.names>
  <animal.subtypes>
   <human id="oid001">
    <human.pets>
     <list-1-n-of-pet_animals>
      <pet_animals>
       <cat-ref entity-instance="oid003"/>
      </pet_animals>
      <pet_animals>
       <cat-ref entity-instance="oid004"/>
      </pet_animals>
      <pet_animals>
       <dog-ref entity-instance="oid005"/>
      </pet_animals>
     </list-1-n-of-pet_animals>
    </human.pets>
    <human.works_for>
```

```
    <human-ref entity-instance="oid002"/>
   </human.works_for>
  </human>
 </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Renee Swank</string>
   </name>
  </list-1-n-of-name>
 </animal.names>
 <animal.subtypes>
  <human id="oid002">
   <human.works_for>
    <human-ref entity-instance="oid006"/>
   </human.works_for>
  </human>
 </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Sigfried</string>
   </name>
  </list-1-n-of-name>
 </animal.names>
 <animal.subtypes>
  <cat id="oid003">
   <cat.addicted_to_catnip>
    <boolean value="true"/>
   </cat.addicted_to_catnip>
  </human>
 </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Bete Noir</string>
   </name>
   <name>
    <string>Beteski</string>
   </name>
  </list-1-n-of-name>
 </animal.names>
 <animal.subtypes>
  <cat id="oid004">
   <cat.addicted_to_catnip>
    <boolean value="true"/>
   </cat.addicted_to_catnip>
  </human>
 </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Forrest</string>
   </name>
   <name>
    <string>Dogboy</string>
```

```
    </name>
    <name>
     <string>Noseboy</string>
    </name>
   </list-1-n-of-name>
  </animal.names>
  <animal.subtypes>
   <dog id="oid005">
    <dog.catches_frisbees>
     <boolean value="false"/>
    </dog.catches_frisbees>
   </human>
  </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Carla Corkern</string>
   </name>
   <name>
    <string>Task Mistress</string>
   </name>
  </list-1-n-of-name>
 </animal.names>
 <animal.subtypes>
  <human id="oid006">
   <human.pets>
    <list-1-n-of-pet_animals>
     <pet_animals>
      <dog-ref entity-instance="oid008"/>
     </pet_animals>
     <pet_animals>
      <dog-ref entity-instance="oid009"/>
     </pet_animals>
    </list-1-n-of-pet_animals>
   </human.pets>
   <human.works_for>
    <human-ref entity-instance="oid007"/>
   </human.works_for>
  </human>
 </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Lucie Feldstat</string>
   </name>
  </list-1-n-of-name>
 </animal.names>
 <animal.subtypes>
  <human id="oid006">
   <human.works_for>
    <human-ref entity-instance="oid007"/>
   </human.works_for>
  </human>
 </animal.subtypes>
</animal>
<animal>
 <animal.names>
  <list-1-n-of-name>
   <name>
    <string>Chauncy</string>
```

```
    </name>
    <name>
     <string>The Chaunce Man</string>
    </name>
   </list-1-n-of-name>
  </animal.names>
  <animal.subtypes>
   <dog id="oid008">
    <dog.catches_frisbees>
     <boolean value="false"/>
    </dog.catches_frisbees>
   </human>
  </animal.subtypes>
 </animal>
 <animal>
  <animal.names>
   <list-1-n-of-name>
    <name>
     <string>Chelsie</string>
    </name>
    <name>
     <string>The Chelster</string>
    </name>
   </list-1-n-of-name>
  </animal.names>
  <animal.subtypes>
   <dog id="oid009">
    <dog.catches_frisbees>
     <boolean value="false"/>
    </dog.catches_frisbees>
   </human>
  </animal.subtypes>
 </animal>
</non-constant-instances>
</people_and_pets>
```

4.2.4. PDML Early Binding Architecture

In addition to the early-binding DTD generation algorithm, the Early Binding workshop also developed an architectural DTD that defines the element type and attribute "templates" that any early-bound document should conform to. Note that this DTD looks very much like a late-bound DTD, which is essentially what it is. The important aspect of this architectural DTD is that the mapping from this late-bound representation to the early bound representation is completely declarative and formal. The mapping itself is done using attributes in the early-bound DTDs.

Another important aspect of using an SGML architecture in this way is that it allows more freedom in how the early-bound documents are represented without impairing interchange because the mapping back to the architecture serves to normalize differences in the different documents into a common vocabulary and syntax.

```
<![IGNORE[====================================

Early Binding Architectural DTD Declarations

This set of declarations defines the general
element types from which schema-specific element
types are derived for early-bound EXPRESS entity
```

```
instance interchange documents.

Version: 0.1

Author: Early Binding Workshop Team

Date: 18 Sept 1999

To Do:

  - Add other metadata attributes
  - Add attribute(s) for binding instances to their schema (for
    use in multi-schema models)
  - Add elements for declaring the schemas used in a multi-schema
    model?


 ============================================<]>

<!--============================================
    Organizing Parameter Entities
    ============================================-->
<!ENTITY % exp-src-name  -- The name of the component as defined in the
original schema --
  "express-source-name
     CDATA
         #REQUIRED"
>

<!ENTITY % constant-atts -- Attributes for labeling constant entity
instances --
  "constant-name -- Name of the entity as a constant, if constant --
     CDATA
     #IMPLIED    -- Has a value only if instance is a constant entity --
   is-constant   -- Is the entity a constant entity instance? --
     (true |
      false)
     false"
>


<!ENTITY % instance -- Entity instances --
  "nested-complex-entity-instance |
   flat-complex-entity-instance"
>

<!ENTITY % entity-instances -- Direct or indirect entity instances --
  "%instance; |
   external-entity-instance-ref"
>

<!ENTITY % ref -- References to entity instances --
  "external-entity-instance-ref |
   entity-instance-ref "
>

<!ENTITY % named-types -- Named data types (includes built-ins --
  "defined-datatype-instance |
   integer|
   real |
   string |
   boolean |
   binary |
   logical |
```

```
   select"
>

<!ENTITY % aggregates -- Aggregate base data types --
  "bag |
   set |
   list |
   array"
>

<!--============================================
    Addressing Notations

    ============================================-->

<!NOTATION xpointer  -- W3C XPointer addresses --
  SYSTEM "http://w3.org/xml/xpointer.xml"
>
<!NOTATION express  -- EXPRESS language schema --
  PUBLIC "ISO 10303-11:1992::NOTATION EXPRESS Language Reference//EN"
>

<!NOTATION unknown-notation
   -- Bridging architecture --
  SYSTEM "{unknown notation}"
>

<!--============================================
    Element Type Declarations
    ============================================-->

<!--============================================
    Document Elements
    ============================================-->

<!ELEMENT model-instance-set  -- A set of entity instances conforming to
a multi-schema model --
  (constant-instances,
   non-constant-instances)
>
<!ATTLIST model-instance-set
  schema-names -- Names of the schemas used in the model --
    CDATA
    #REQUIRED
  hub-schema   -- The schema that definition of the model starts with --
    CDATA
    #REQUIRED
  id
    ID
    #IMPLIED
  express-schemas  -- Pointer to actual EXPRESS schema file --
    ENTITIES       -- External data entity with a notation of "express" -
-
    #IMPLIED       -- Specify only if schema exists in EXPRESS form --

>

<!ELEMENT schema-instance-set  -- A set of entity instances conforming to
a single schema --
  (constant-instances,
   non-constant-instances)
>
<!ATTLIST schema-instance-set
  schema-name -- Name of the schema as declared on SCHEMA declaration --
```

```
    CDATA
    #REQUIRED
  id
    ID
    #IMPLIED
  express-schema  -- Pointer to actual EXPRESS schema file --
    ENTITY        -- External data entity with a notation of "express" --
    #IMPLIED      -- Specify only if schema exists in EXPRESS form --
>

<!--================================================
    Organizing Containers
    ================================================-->
<!ELEMENT constant-instances -- Constant instances declared in schema --
  (%entity-instances;)*
>

<!ELEMENT non-constant-instances -- Instances not declared as constants
in the schema --
  (%entity-instances;)*
>

<!--================================================
    Entity Instances
    ================================================-->

<!ELEMENT flat-complex-entity-instance -- An entity instance which has
all its subtypes as its direct children. --
  -- Must be used for entity types in a graph that contains
     multiple supertypes --
  (external-partial-entity-instance-ref |
   partial-entity-instance)+
>
<!ATTLIST flat-complex-entity-instance
   %exp-src-name;
   id
     ID
         #IMPLIED
   %constant-atts;
>

<!ELEMENT nested-complex-entity-instance -- An entity that is the root of
a supertype tree (and possibly the leaf of that tree) --
  (attribute*,
   subtypes)
>
<!ATTLIST nested-complex-entity-instance
  %exp-src-name;
  id
     ID
         #IMPLIED
   %constant-atts;
>

<!ELEMENT subtypes -- Container for subtype entity instances of a singly-
supertyped-entity-instance --
  (nested-partial-entity-instance |
   external-entity-instance-ref)*
>
<!ELEMENT nested-partial-entity-instance -- An entity that is an
intermediate or leaf partial entity instance in a
singly-rooted supertype tree) --
  (attribute*,
   subtypes?)
```

```
>
<!ATTLIST nested-partial-entity-instance
  %exp-src-name;
  id
     ID
         #IMPLIED
   %constant-atts;
>

<!ELEMENT entity-instance-ref -- A reference to a complex or partial
entity instance in the same model population --
  EMPTY
>
<!ATTLIST entity-instance-ref
  entity-instance
    CDATA
    #REQUIRED
  id
     ID
         #IMPLIED
  -- HyTime attributes defining this as a use-by-reference --
  valueref -- This element is a redirect to the target element --
    CDATA
    #FIXED "#ELEMENT p-e-i"
  loctype  -- Must specify in the instance the addressing method used.  -
-
    CDATA
    #REQUIRED
  reftype
    CDATA
    #FIXED "entity-instance (partial-entity-instance | nested-partial-
entity-instance)"
  HyTime -- Mapped to generic form in HyTime arch --
    NAME
    #FIXED "HyBrid"
>

<!ELEMENT partial-entity-instance -- A partial entity instance component
of a complex entity instance --
  (attribute*)
>
<!ATTLIST partial-entity-instance
  %exp-src-name;
  id
     ID
         #IMPLIED
>

<!ELEMENT attribute -- An attribute of an entity instance --
  (%ref; |
   %instance; |
   %named-types; |
   %aggregates;)
>
<!ATTLIST attribute
  %exp-src-name;
>

<!--============================================
    Data Types
    ============================================-->

<!ELEMENT defined-datatype-instance -- An instance of a defined non-
entity data type --
```

```
  (%named-types; |
   %aggregates; |
   %ref; |
   %instance;)
>
<!ATTLIST defined-datatype-instance
  %exp-src-name;
>

<!ELEMENT select -- An instance of a select type --
  (%named-types;)
>
<!ATTLIST select
  %exp-src-name;
>

<!ELEMENT (integer |real | string | boolean | binary | logical)
  (#PCDATA)
>

<!ELEMENT (bag | list | set | array)
  ((%named-types;)* |
   (unnamed-type)*)
>
<!ATTLIST (bag | list | set)
  upper-bound
    CDATA
    #REQUIRED
  lower-bound
    CDATA
    #REQUIRED
>
<!-- %named-types-decl; -->

<!ELEMENT unnamed-type -- An unnamed aggregate type (e.g., set of list of
real) --
  ((%named-types;)* |
   (unnamed-type)*)
>
```

4.2.5. Critique of PDML Early Binding

The PDML Early Binding, as defined as of the end of the second Early Binding workshop, provides a completely satisfactory early-bound representation of EXPRESS-driven data. It reduces the markup needed to the absolute minimum without sacrificing completeness.

As I had a significant influence on the ultimate design of the PDML Early Binding design, it should be no surprise that I consider it to not have any examples of poor XML design.

4.3. Part 28 Late Binding (ISO TC184/SC4)

The Part 28 Late Binding approach has the goal of satisfying, at a minimum, the requirements for character-based interchange of EXPRESS entity instances currently satisfied by Part 21. This work is being conducted as a New Work Item managed under WG11 of TC184/SC4. Until recently, progress was slow because the effort was entirely

volunteer until the British Standards Institute funded development of the first draft of the Part 28 specification.

An XML markup approach for EXPRESS data was first discussed in any detail at the STEP meeting in Orlando, Florida in January 1998, when the author informally demonstrated one of many possible approaches to using XML for representing EXPRESS data. Between that meeting and the following meeting in Bad Aibling, Germany, Daniel Rivers-Moore, then co-chair of the SGML and Industrial Data preliminary work item (PWI), developed an example markup approach and worked with several product vendors to develop demonstrations of XML-based import and export. These demonstrations were presented formally in Bad Aibling. Also at the Bad Aibling meeting, a set of design principles was developed and a requirements document was started.

Little additional concrete technical progress was made until the STEP meeting in San Francisco in January 1999, when the PWI group drafted a late-bound representation design with a focus on representing EXPRESS schemas, in particular, using XML to represent expressions within schemas. Between the San Francisco meeting and the Lillehammer meeting in June, the XML representation new work item (NWI) was approved, so most of the time in Lillehammer was spent on discussions of the NWI, various technology options (including XMI and PDML), and so on.

Thus, the Part 28 Early Binding specification is the least complete of the three and the one most likely to change in reaction to new design insights, new requirements, and pressure from forces outside the STEP community.

4.3.1. Part 28 Schema Instance DTD

This DTD is an early draft of the Part 28 schema instance document type. It is early bound to the schema structures, but is used to include schemas with late-bound interchange packages (it could also be used with early-bound interchange packages as long as there are no element type conflicts). The DTD presented here does not necessarily reflect the latest work done on the Part 28 DTD, but is is representative of the Part 28 approach.

```
<![IGNORE[-------------------------------------------------

   Example XML Representation DTD

   This DTD reflects the application of the design principles
   and requirements developed at the San Francisco meeting
   to the problem of using XML to represent EXPRESS schemas
   and data instances.  It is simply a design exercise--it
   does not represent any sort of formal proposal or submission.

  $Header:
/data/cvsroot/customer/nist/documents/exprxmlanal/exprxmlanal.sgm,v 1.17
1999/10/21 17:17:57 eliot Exp $
```

```
  ------------------------------------------------------------<]>

<!ELEMENT express-driven-data
  (schema,
   data)
>

<!--=======================================================
    Schema Representation
        =======================================================-->

<!ELEMENT schema
  (imports,
   constant_block,
   (entity_decl |
    function_decl |
    procedure_decl |
    rule_decl |
    type_decl)+)
>
<!ATTLIST schema
  id
    ID
    #REQUIRED
>

<!ELEMENT imports
  (#PCDATA)*
>

<!ELEMENT constant_block
  (#PCDATA)*
>

<!ELEMENT type_decl
  (type_name,
   (base_type |
    select))
>

<!ELEMENT type_name
  (#PCDATA)*
>

<!ELEMENT select
  (type_ref,
   type_ref+)
>
<!ELEMENT type_ref
  (#PCDATA)*
>
<!ELEMENT entity_decl
  (entity_id,
   supertype_constraint?,
   subtype_decl?,
   attribute_def*,
   unique_rule?,
   where_rules?)
>

<!ELEMENT supertype_constraint
  (abstract_supertype |
   andor)
>
```

```
<!ELEMENT abstract_supertype
  (andor?)
>

<!ELEMENT andor
  (supertype_factor,
   supertype_factor+)
>

<!ELEMENT supertype_factor
  ((entity_ref |
    one_of |
    andor),
   (entity_ref |
       one_of |
    andor)+)
>

<!ELEMENT one_of
  (andor,
   andor+)
>

<!ELEMENT subtype_decl
  (entity_ref+)
>

<!ELEMENT entity_id
  (#PCDATA)*
>

<!ELEMENT attribute_def
  (attribute_name,
   optional?,
   attribute_type)
>

<!ELEMENT attribute_name
  (#PCDATA)*
>

<!ELEMENT optional
  EMPTY
>

<!ELEMENT attribute_type
  (string |
   integer |
   real |
   boolean |
   logical |
   list |
   set |
   bag |
   enum |
   type_ref |
   entity_ref)

>

<!ELEMENT function_decl
  (function_id,
   arguments,
```

```
   algorithm_head?,
   return_type,
   expression)
>

<!ELEMENT function_id
  (#PCDATA)*
>

<!ELEMENT arguments
  (formal_parameter*)
>

<!ELEMENT formal_parameter
  (parameter_id+,
   parameter_type)
>

<!ELEMENT parameter_id
  (#PCDATA)*
>

<!ELEMENT parameter_type
  (string |
   integer |
   real |
   list |
   set |
   bag |
   enum |
   type_ref |
   entity_ref)
>

<!ELEMENT algorithm_head
  ((entity_decl |
    function_decl |
    procedure_decl |
    type_decl),
   constant_decl?,
   local_decl?)
>

<!ELEMENT return_type
  (string |
   integer |
   real |
   list |
   set |
   bag |
   enum |
   type_ref |
   entity_ref)
>

<!ELEMENT constant_decl
  (constant_id,
   base_type,
   expression)
>

<!ELEMENT constant_id
  (#PCDATA)*
>
```

```
<!ELEMENT base_type
  (string |
   integer |
   real |
   list |
   set |
   bag |
   enum |
   type_ref |
   entity_ref)
>

<!ELEMENT local_decl
  (variable_id+,
   parameter_type,
   expression)
>

<!ELEMENT procedure_decl
  (function_id,
   arguments,
   algorithm_head?,
   expression)
>

<!ELEMENT rule_decl
  (rule_id,
   entity_ref+,
   algorithm_head?,
   expression,
   where_clause?)
>

<!ELEMENT rule_id
  (#PCDATA)*
>

<!ELEMENT where_clause
  (domain_rule+)
>

<!ELEMENT domain_rule
  (label,
   expression)
>

<!ELEMENT string
  EMPTY
>
<!ELEMENT integer
  EMPTY
>
<!ELEMENT boolean
  EMPTY
>
<!ELEMENT logical
  EMPTY
>
<!ELEMENT real
  (precision?)
>

<!ELEMENT precision
```

```
  (#PCDATA)
>

<!ELEMENT array
  (cardinality,
   allowed_types)
>

<!ELEMENT list
  (cardinality?,
   unique?,
   allowed_types)
>

<!ELEMENT cardinality
  (min,
   max)
>

<!ELEMENT min
  (#PCDATA)*
>
<!ELEMENT max
  (#PCDATA)*
>

<!ELEMENT unique
  EMPTY
>

<!ELEMENT set
  (cardinality?,
   unique?,
   allowed_types)
>

<!ELEMENT bag
  (cardinality?,
   unique?,
   allowed_types)
>

<!ELEMENT enum
  (member+)
>

<!ELEMENT member
  (#PCDATA)
>

<!ELEMENT allowed_types
  (type_ref*)
>

<!ELEMENT unique_rule
  (attribute_name_ref*)
>

<!ELEMENT label
  (#PCDATA)*
>

<!ELEMENT where_rules
  (where_rule*)
```

```
>
<!ELEMENT where_rule
  (label?,
   expression)
>

<!--=======================================
    Data Instance Representation
        =======================================-->

<!ELEMENT data
  (entity_instance*)
>

<!ELEMENT entity_instance
  (partial_entity_instance*)
>
<!ATTLIST entity_instance
  id
     ID
     #REQUIRED
>

<!ELEMENT partial_entity_instance
  (entity_ref,
   attribute*)
>

<!ELEMENT entity_ref
  (id_val |
   (self |
    variable_ref |
    expression))
>

<!ELEMENT id_val
  (#PCDATA)*
>

<!ELEMENT attribute
  (attribute_name_ref,
   (select_type_path |
    type_ref),
   (simple_value |
    entity_relationship))
>

<!ELEMENT attribute_name_ref
  (#PCDATA)*
>

<!ELEMENT select_type_path
  (type_ref+)
>

<!ELEMENT simple_value
  (enum_val |
   string_val |
   integer_val |
   real_val |
   list_val |
   set_val |
   bag_val |
   binary_val |
```

```
   logical_val |
   boolean_val)
>

<!ELEMENT string_val
  (#PCDATA)*
>

<!ELEMENT boolean_val
  (true | false)
>
<!ELEMENT logical_val
  (true | false | unknown)
>

<!ELEMENT integer_val
  (#PCDATA)*
>
<!ELEMENT real_val
  (rlbase,
   rlmantissa)
>
<!ELEMENT rlbase
  (#PCDATA)*
>
<!ELEMENT rlmantissa
  (#PCDATA)*
>

<!NOTATION uuencode PUBLIC "uuencoded data" >
<!NOTATION mime     PUBLIC "mime encoded data" >
<!NOTATION base64   PUBLIC "base-64 encoded data" >

<!ELEMENT binary_val
  (#PCDATA)*
>
<!ATTLIST binary_val
  notation
    NOTATION
    (uuencode |
     mime |
     base64)
    "uuencode"
>

<!ELEMENT set_val
  (string_val* |
   integer_val* |
   real_val* |
   binary_val* |
   set_val* |
   list_val* |
   bag_val* |
   enum_val*)
>

<!ELEMENT list_val
  (string_val* |
   integer_val* |
   real_val* |
   binary_val* |
   set_val* |
   list_val* |
   bag_val* |
```

```
   enum_val*)
>

<!ELEMENT bag_val
  (string_val* |
   integer_val* |
   real_val* |
   binary_val* |
   set_val* |
   list_val* |
   bag_val* |
   enum_val*)
>

<!ELEMENT enum_val
  (entity_instance_ref?,
   attribute_ref?,
   member_ref)
>

<!ELEMENT attribute_ref
  ((entity_ref |
    self)?,
    id_val,
    attribute_ref*)
>

<!ELEMENT member_ref
  (#PCDATA)*
>

<!ELEMENT entity_relationship
  (entity_instance_ref*)
>

<!ELEMENT entity_instance_ref
  (#PCDATA)*
>

<!--=======================================================
    Expression Element types
    =======================================================-->

<!ENTITY % expression_components
  "in |
   like |
   less_than |
   greater_than |
   less_than_or_equal |
   greater_than_or_equal |
   not_equal |
   equal |
   instance_equal |
   instance_not_equal |
   negate |
   identity |
   not |
   add |
   subtract |
   mult |
   real_divide |
   integer_divide |
   modulo |
   exponentiate |
```

```
    and |
    or |
    xor |
    intersection |
    union |
    difference |
    subset |
    superset |
    construct |
    abs |
    acos |
    asin |
    atan |
    blength |
    cos |
    exists |
    exp |
    format |
    hibound |
    hiindex |
    length |
    lobound |
    loindex |
    log |
    log2 |
    log10 |
    nvl |
    odd |
    rolesof |
    query |
    sin |
    sizeof |
    sqrt |
    tan |
    typeof |
    usedin |
    value |
    value_in |
    value_unique |
    alias_stmt |
    assignment_stmt |
    case_stmt |
    compound_stmt |
    escape_stmt |
    if_stmt |
    null_stmt |
    procedure_call_stmt |
    repeat_stmt |
    return_stmt |
    skip_stmt |
    insert |
    remove |
    const_e |
    indedeterminate |
    false |
    true |
    pi |
    self |
    unknown |
    index |
    concatenation |
    aggregate_initializer"
>
```

```
<!ELEMENT expression
  (%expression_components;)*
>

<!ELEMENT arg
  (string_val |
   integer_val |
   real_val |
   list_val |
   set_val |
   bag_val |
   binary_val |
   entity_instance_ref |
   attribute_ref |
   variable_ref |
   %expression_components; |
   expression)
>

<!ELEMENT alias_stmt
   (variable_id,
    (parameter_ref |
     variable_ref),
    expression+)
>

<!ELEMENT variable_id
  (#PCDATA)*
>

<!ELEMENT variable_ref
  (#PCDATA)*
>

<!ELEMENT parameter_ref
  (#PCDATA)*
>

<!ELEMENT assignment_stmt
   ((parameter_ref |
     variable_ref),
    expression)
>

<!ELEMENT case_stmt
   (expression,
    case_action+,
    otherwise?)
>

<!ELEMENT case_action
  (case_label+,
   expression)
>

<!ELEMENT case_label
  (expression+)
>

<!ELEMENT otherwise
  (expression)
>

<!ELEMENT compound_stmt
```

```
   (expression+)
>

<!ELEMENT escape_stmt
   EMPTY
>

<!ELEMENT if_stmt
   (expression,
    then,
    else?)
>

<!ELEMENT then
  (expression+)
>
<!ELEMENT else
  (expression+)
>

<!ELEMENT null_stmt
   (arg+)
>

<!ELEMENT procedure_call_stmt
   (procedure_ref,
    arg*)
>

<!ELEMENT procedure_ref
  (#PCDATA)*
>

<!ELEMENT repeat_stmt
   ((repeat_control |
     while_control |
     until_control),
    expression+
   )
>

<!ELEMENT repeat_control
  (variable_id,
   bound_1,
   bound_2,
   by_increment?)
>
<!ELEMENT bound_1
  (#PCDATA)*
>
<!ELEMENT bound_2
  (#PCDATA)*
>
<!ELEMENT by_increment
  (#PCDATA)*
>

<!ELEMENT while_control
  (expression)
>
<!ELEMENT until_control
  (expression)
>
```

```
<!ELEMENT return_stmt
   (expression)
>
<!ELEMENT skip_stmt
   EMPTY
>

<!ELEMENT query
  (variable_id,
   aggregate_source,
   expression)
>

<!ELEMENT aggregate_source
  (attribute_ref |
   expression)
>

<!ELEMENT abs
  (arg)
>

<!ELEMENT acos
  (arg)
>

<!ELEMENT asin
  (arg)
>

<!ELEMENT atan
   (arg,
    arg)
>
<!ELEMENT blength
   (arg)
>
<!ELEMENT cos
   (arg)
>
<!ELEMENT exists
   (arg)
>
<!ELEMENT exp
   (arg)
>
<!ELEMENT format
   (arg,
    string)
>
<!ELEMENT hibound
   (arg)
>
<!ELEMENT hiindex
   (arg)
>
<!ELEMENT length
   (arg)
>
<!ELEMENT lobound
   (arg)
>
<!ELEMENT loindex
   (arg)
```

```
>
<!ELEMENT log
   (arg)
>
<!ELEMENT log2
   (arg)
>
<!ELEMENT log10
   (arg)
>
<!ELEMENT nvl
   (arg,
    arg)
>
<!ELEMENT odd
   (arg)
>
<!ELEMENT rolesof
   (arg)
>
<!ELEMENT sin
   (arg)
>
<!ELEMENT sizeof
   (arg)
>
<!ELEMENT sqrt
   (arg)
>
<!ELEMENT tan
   (arg)
>
<!ELEMENT typeof
   (arg)
>
<!ELEMENT usedin
   (arg,
    arg)
>
<!ELEMENT value
   (arg)
>
<!ELEMENT value_in
   (arg,
    arg)
>
<!ELEMENT value_unique
   (arg)
>

<!ELEMENT insert
  (variable_ref,
   arg,
   arg)
>

<!ELEMENT remove
  (variable_ref,
   arg)
>

<!ELEMENT const_e
  EMPTY
>
```

```
<!ELEMENT indedeterminate
  EMPTY
>

<!ELEMENT false
  EMPTY
>

<!ELEMENT true
  EMPTY
>
<!ELEMENT pi
  EMPTY
>
<!ELEMENT self
  EMPTY
>
<!ELEMENT unknown
  EMPTY
>

<!ELEMENT in
  (arg,
   arg)
>
<!ELEMENT like
  (arg,
   arg)
>
<!ELEMENT less_than
  (arg+)
>
<!ELEMENT greater_than
  (arg+)
>
<!ELEMENT less_than_or_equal
  (arg+)
>
<!ELEMENT greater_than_or_equal
  (arg+)
>
<!ELEMENT not_equal
  (arg+)
>
<!ELEMENT equal
  (arg+)
>
<!ELEMENT instance_equal
  (arg+)
>
<!ELEMENT instance_not_equal
  (arg+)
>
<!ELEMENT negate
  (arg)
>
<!ELEMENT identity
  (arg)
>
<!ELEMENT not
  (arg+)
>
<!ELEMENT add
```

```
  (arg+)
>
<!ELEMENT subtract
  (arg,
   arg)
>
<!ELEMENT mult
  (arg+)
>
<!ELEMENT real_divide
  (arg,
   arg)
>
<!ELEMENT integer_divide
  (arg,
   arg)
>
<!ELEMENT modulo
  (arg,
   arg)
>
<!ELEMENT exponentiate
  (arg,
   arg)
>
<!ELEMENT and
  (arg+)
>
<!ELEMENT or
  (arg+)
>
<!ELEMENT xor
  (arg,
   arg)
>
<!ELEMENT construct
  (arg+)
>
<!ELEMENT index
  (variable_ref,
   arg,
   arg)
>

<!ELEMENT concatenation
  (arg,
   arg+)
>

<!ELEMENT intersection
  (arg,
   arg+)
>

<!ELEMENT union
  (arg,
   arg+)
>

<!ELEMENT difference
  (arg,
   arg+)
>
```

```
<!ELEMENT subset
  (arg,
   arg)
>

<!ELEMENT superset
  (arg,
   arg)
>

<!ELEMENT aggregate_initializer
  (variable_ref?,
   arg+)
>

<![IGNORE[============================================================

  $Log: exprxmlanal.sgm,v $
  Revision 1.17  1999/10/21 17:17:57  eliot
  Fixed name of NIST

  Revision 1.16  1999/10/16 18:45:01  eliot
  Finished editorial work

  Revision 1.15  1999/10/16 16:46:23  eliot
  Wrote glossary

  Revision 1.14  1999/10/16 15:52:54  eliot
  Added section on XML Schema

  Revision 1.13  1999/10/14 05:05:03  eliot
  Did editorial pass
  Added verbiage

  Revision 1.12  1999/10/14 03:48:16  eliot
  Added part 28 examples

  Revision 1.9  1999/07/23 03:17:08  eliot
  Added logical_val and boolean_val

  Revision 1.8  1999/06/07 08:58:26  eliot
  Fixed notation attribute for binary_val


  Revision 1.7  1999/06/07 07:24:23  eliot
  Added RCS log

  Revision 1.6  1999/06/07 07:23:43  eliot
  Latest tweaks

  Revision 1.5  1999/01/29 00:45:32  eliot
  Validated and tested with sample doc. Still not verified or reviewed.

  Revision 1.4  1999/01/28 22:49:31  eliot
  Most expressions and statements coded up. Validates but not verified

  Revision 1.3  1999/01/28 21:53:14  eliot
  Added header

  ====================================================================<]>
```

4.3.2. Part 28 Late Bound Data Instance DTD

This DTD is the most recent draft of the Part 28 data instance DTD, provided by the project editor, Robin La Fontaine. It does not necessarily reflect the final design of the Part 28 Late Binding DTD.

```
<!-- Late binding data section of XML based on Eliot’s original.
Suitably improved/corrupted by Robin La Fontaine
Version 1, 7 Sept 1999

Version 2, 11 October 1999: Updated using Eliot Kimber’s architectural
DTD
for late-bound data which was developed at the Early-binding workshop at
Long Beach, CA,
in September 1999.

-->

<!--============================================
    Items brought over from main language schema
    ============================================
(I know there are other ways to do this, but this is the simplest
for the software I am using!) -->

<!ENTITY % literal ’ binary_literal |
  integer_literal |
  boolean_literal |
  logical_literal |
  real_literal |
  string_literal ’>

<!ELEMENT entity_ref (#PCDATA)>
<!ELEMENT attribute_ref (#PCDATA)>
<!ELEMENT schema_ref (#PCDATA)>
<!ELEMENT type_ref (#PCDATA)>
<!ELEMENT enumeration_ref (#PCDATA)>
<!ELEMENT string_literal (#PCDATA)>
<!ELEMENT integer_literal (#PCDATA)>
<!ELEMENT real_literal (#PCDATA)>
<!ELEMENT binary_literal (#PCDATA)>
<!ELEMENT boolean_literal (false | true)> <!-- WEK: Added boolean_literal
-->
<!ELEMENT logical_literal (false | true | unknown)>
<!ELEMENT false EMPTY>
<!ELEMENT true EMPTY>
<!ELEMENT unknown EMPTY>

<!-- data is brought over and modified here -->
<!ELEMENT data
  (data_id, schema_instance)
>

<!--============================================
    Organizing Parameter Entities
    ============================================-->

<!ENTITY % entity_instance
  "nested_complex_entity_instance |
   flat_complex_entity_instance"
>

<!ENTITY % entity_instance_ref
```

```
  "flat_complex_entity_instance_ref |
   partial_entity_instance_ref |
   nested_complex_entity_instance_ref |
   nested_complex_entity_instance_subtype_ref"
>

<!ENTITY % simple_value
  "%literal; |
   bag_literal |
   list_literal |
   set_literal |
   array_literal |
   type_literal |
   entity_literal"
>

<!ENTITY % aggregate-contents
  "binary_literal* |
   integer_literal* |
   logical_literal* |
   real_literal* |
   string_literal* |

   bag_literal* |
   list_literal* |
   set_literal* |
   array_literal* |

   type_literal* |
   entity_literal*"
>

<!--================================================
    Main section of DTD
    ================================================-->

<!-- If we have multiple data sets, each should have a unique identifier.
-->
<!ELEMENT data_id
  (#PCDATA)
>

<!ELEMENT schema_instance
  (schema_ref,
   constant_instances,
   non_constant_instances)
>

<!ELEMENT constant_instances
  (%entity_instance;)*
>

<!ELEMENT non_constant_instances
  (%entity_instance;)*
>

<!ELEMENT flat_complex_entity_instance
  (entity_instance_id, partial_entity_instance+)
>

<!ELEMENT nested_complex_entity_instance
  (entity_instance_id,
   (entity_ref | external_entity_ref),
   attribute_instance*,
```

```
   nested_complex_entity_instance_subtype*)
>

<!ELEMENT nested_complex_entity_instance_subtype
  (entity_instance_id?,
   (entity_ref | external_entity_ref),
   attribute_instance*,
   nested_complex_entity_instance_subtype*)
>

<!-- If we use an attribute of type ID we will need to make it unique to
the
entire file, but with this it can be unique to a data set. Also we avoid
use
of
attributes if possible. -->
<!ELEMENT entity_instance_id
  (#PCDATA)
>

<!ELEMENT partial_entity_instance
  (entity_instance_id?,
   (entity_ref | external_entity_ref),
   attribute_instance*)
>

<!-- external_entity_ref needed when there are multiple
     schemas in the interchange package. If absent, then the item is in
the
schema
which is referenced as the main schema within schema_instance.
 -->
<!ELEMENT external_entity_ref
  (entity_ref, schema_ref)
>

<!ELEMENT attribute_instance
  (attribute_ref,
   (%simple_value;))
>

<!ELEMENT external_type_ref
  (type_ref, schema_ref)
>


<!NOTATION uuencode PUBLIC "uuencoded data" >
<!NOTATION mime     PUBLIC "mime encoded data" >
<!NOTATION base64   PUBLIC "base-64 encoded data" >

<!ATTLIST binary_literal
  notation
    NOTATION
    (uuencode |
     mime |
     base64)
    "uuencode"
>

<!ELEMENT set_literal
  (%aggregate-contents;)
>

<!ELEMENT list_literal
```

```
  (%aggregate-contents;)
>

<!ELEMENT bag_literal
  (%aggregate-contents;)
>

<!ELEMENT array_literal
  (%aggregate-contents;)
>

<!ELEMENT type_literal
  ((type_ref | external_type_ref),
   (%simple_value; | enumeration_ref)
    )
>

<!ELEMENT entity_literal
  (%entity_instance; | %entity_instance_ref;)
>

<!ELEMENT flat_complex_entity_instance_ref
  (#PCDATA)
>

<!ELEMENT partial_entity_instance_ref
  (#PCDATA)
>

<!ELEMENT nested_complex_entity_instance_ref
  (#PCDATA)
>

<!ELEMENT nested_complex_entity_instance_subtype_ref
  (#PCDATA)
>
```

4.3.3. Part 28 Schema Instance

The following is the people and pets schema defined using the Part 28 schema DTD.

```
<!DOCTYPE express-driven-data SYSTEM "p2x0_0.dtd">
<express-driven-data>
 <schema>
  <imports/>
  <constant_block/>
  <type_decl>
   <type_name>name</type_name>
   <base_type>
    <string/>
   </base_type>
  </type_decl>
  <type_decl>
   <type_name>pet_animals</type_name>
   <select>
    <type_ref>dog</type_ref>
    <type_ref>cat</type_ref>
   </select>
  </type_decl>
  <entity_decl>
   <entity_id>animal</entity_id>
   <supertype_constraint>
```

```
   <andor>
    <supertype_factor>
     <entity_ref>
      <id_val>human</id_val>
     </entity_ref>
    </supertype_factor>
    <supertype_factor>
     <entity_ref>
      <id_val>dog</id_val>
     </entity_ref>
    </supertype_factor>
    <supertype_factor>
     <entity_ref>
      <id_val>cat</id_val>
     </entity_ref>
    </supertype_factor>
   </andor>
  </supertype_constraint>
  <attribute_def>
   <attribute_name>names</attribute_name>
   <attribute_type>
    <list>
     <cardinality>
      <min>1</min>
      <max>*</max>
     </cardinality>
     <allowed_types>
      <type_ref>name</type_ref>
     </allowed_types>
    </list>
   </attribute_type>
  </attribute_def>
 </entity_decl>
 <entity_decl>
  <entity_id>human</entity_id>
  <subtype_decl>
   <entity_ref>
    <id_val>animal</id_val>
   </entity_ref>
  </subtype_decl>
  <attribute_def>
   <attribute_name>works_for</attribute_name>
   <attribute_type>
    <entity_ref>
     <id_val>human</id_val>
    </entity_ref>
   </attribute_type>
  </attribute_def>
  <attribute_def>
   <attribute_name>pets</attribute_name>
   <optional/>
   <attribute_type>
    <list>
     <cardinality>
      <min>1</min>
      <max>*</max>
     </cardinality>
     <allowed_types>
      <type_ref>pet_animals</type_ref>
     </allowed_types>
    </list>
   </attribute_type>
  </attribute_def>
 </entity_decl>
```

```
  <entity_decl>
   <entity_id>dog</entity_id>
   <subtype_decl>
    <entity_ref>
     <id_val>animal</id_val>
    </entity_ref>
   </subtype_decl>
   <attribute_def>
    <attribute_name>catches_frisbees</attribute_name>
    <attribute_type>
     <boolean/>
    </attribute_type>
   </entity_decl>
   <entity_decl>
    <entity_id>cat</entity_id>
    <subtype_decl>
     <entity_ref>
      <id_val>animal</id_val>
     </entity_ref>
    </subtype_decl>
    <attribute_def>
     <attribute_name>addicted_to_catnip</attribute_name>
     <attribute_type>
      <boolean/>
     </attribute_type>
    </attribute_def>
  </entity_decl>
 </schema>
</express-driven-data>
```

4.3.4. Part 28 Late Binding Data Instance

The following is a data instance document conforming to the Part 28 Late Bound data instance DTD.

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "p28-eb.dtd">
<data>
 <data_id>oid000</data_id>
 <schema_instance>
  <schema_ref>p28-pnp-schema.xml</schema_ref>
  <constant_instances>
  </constant_instances>
  <non_constant_instances>
   <nested_complex_entity_instance>
    <entity_instance_id>oid000</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>W. Eliot Kimber</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid001</entity_instance_id>
     <entity_ref>human</entity_ref>
     <attribute_instance>
      <attribute_ref>works_for</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid002</partial_entity_instance_ref>
      </entity_literal>
```

```
     </attribute_instance>
     <attribute_instance>
      <attribute_ref>cat</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid003</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
     <attribute_instance>
      <attribute_ref>cat</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid004</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
     <attribute_instance>
      <attribute_ref>dog</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid005</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid0010</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Renee Swank</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid002</entity_instance_id>
     <entity_ref>human</entity_ref>
     <attribute_instance>
      <attribute_ref>works_for</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid006</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid0011</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Sigfried</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid003</entity_instance_id>
     <entity_ref>cat</entity_ref>
     <attribute_instance>
      <attribute_ref>addicted_to_catnip</attribute_ref>
      <boolean_literal><true/></boolean_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
```

```
    <entity_instance_id>oid0012</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Bete Noir</string_literal>
      <string_literal>Beteski</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid004</entity_instance_id>
     <entity_ref>cat</entity_ref>
     <attribute_instance>
      <attribute_ref>addicted_to_catnip</attribute_ref>
      <boolean_literal><true/></boolean_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid0013</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Forrest</string_literal>
      <string_literal>Dogboy</string_literal>
      <string_literal>Noseboy</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid005</entity_instance_id>
     <entity_ref>dog</entity_ref>
     <attribute_instance>
      <attribute_ref>chases_frisbees</attribute_ref>
      <boolean_literal><false/></boolean_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid014</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Carla Corkern</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid006</entity_instance_id>
     <entity_ref>human</entity_ref>
     <attribute_instance>
      <attribute_ref>works_for</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid007</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
     <attribute_instance>
      <attribute_ref>cat</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid008</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
```

```
     <attribute_instance>
      <attribute_ref>cat</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid009</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid0015</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Lucie Feldstad</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid007</entity_instance_id>
     <entity_ref>human</entity_ref>
     <attribute_instance>
      <attribute_ref>works_for</attribute_ref>
      <entity_literal>
       <partial_entity_instance_ref>pointer to
oid007</partial_entity_instance_ref>
      </entity_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid0016</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Chauncy</string_literal>
      <string_literal>The Chaunce Man</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid008</entity_instance_id>
     <entity_ref>dog</entity_ref>
     <attribute_instance>
      <attribute_ref>chases_frisbees</attribute_ref>
      <boolean_literal><false/></boolean_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
   <nested_complex_entity_instance>
    <entity_instance_id>oid0017</entity_instance_id>
    <entity_ref>animal</entity_ref>
    <attribute_instance>
     <attribute_ref>names</attribute_ref>
     <list_literal>
      <string_literal>Chelsie</string_literal>
      <string_literal>The Chelster</string_literal>
     </list_literal>
    </attribute_instance>
    <nested_complex_entity_instance_subtype>
     <entity_instance_id>oid009</entity_instance_id>
     <entity_ref>dog</entity_ref>
     <attribute_instance>
      <attribute_ref>chases_frisbees</attribute_ref>
```

```
      <boolean_literal><false/></boolean_literal>
     </attribute_instance>
    </nested_complex_entity_instance_subtype>
   </nested_complex_entity_instance>
  </non_constant_instances>
 </schema_instance>
</data>
```

4.3.5. Critique of Part 28 Late Binding

It should be noted that the Part 28 Late Binding presented here and the PDML Early Binding architecture presented earlier are quite similar. This is no accident, as one of the reasons for creating the early binding architecture was to see if such an architecture could also be used directly for late bound instances. The two DTDs do exhibit some differences, stemming mostly from the original design decision to always represent complex entity instances as flat sets of partial entity instances with no nesting to represent subtype/supertype relationships or containment of entities in the attribute values of other entities. However, this design decision was simply a reflection of a desire for simplicity (do it one way), rather than a necessary aspect of a good late binding design.

The schema portion of the DTD reflects the syntax of the EXPRESS language as defined in ISO 10303 Part 11. This was a conscious design decision to reflect the terminology and names used in the EXPRESS syntax productions. The rational was that these names were familiar to the target audience and that it would enable easier inspection of the final design for correctness. My personal preference at the moment is to use markup that reflects the abstraction of the schema rather than its original syntactic representation. Of course, lacking a defined and standardized abstract semantic model for EXPRESS, the only thing you can define with certainty is a purely syntactic mapping.

In the design shown here, the mechanism by which entity instances point to entity types is underspecified, although the intent was, I believe, that pointer values would be XPath pointers.

Glossary of Technical Terms

As a convenience, this glossary provides informal or paraphrased definitions of the EXPRESS and XML jargon terms used in this report. Please see the official standards documents for the complete and formal definitions of these terms.

attribute

In EXPRESS, a property of an entity. Attribute shave values that are either simple types, aggregations of simple types, entity instances, or aggregations of entity instances.

In XML, a property of an element, syntactically specified as a string value within the start tag for the element.

complex entity instance

In EXPRESS, the representation of a single entity. A complex entity instance contains one or more partial entity instances, one for each distinct entity type that governs the complex entity instance.

content

In XML, the character data and/or elements syntactically contained within another element. Semantically, the tree of elements within an XML document is the "content" of the document (as distinct from the other parts of the document, such as the document type declaration).

document

In XML, a character string that conforms to the production for the non-terminal "document" in the XML 1.0 specification. XML distinguishes documents from document fragments. Abstractly, an XML document is an object that binds a tree of elements to a set of syntactic constraints (the document type declaration or DTD).

document instance

The tree of elements within an XML document. Informally, people often use the term "document instance" to mean the entire XML document.

document type

In XML, the syntax constraints that govern the elements in an XML document.

Note that the name "document *type*" is a bit of a misnomer as a document type governs exactly one document, namely the document that it is a part of, and does not (and cannot) define a true "type" in the sense of a set of

rules that can be applied to multiple data instances. In practice, people use the syntactic trick of external declaration sets to get more or less the effect of true types, but this practice is not reliable or formalized.

early bound

For representation syntaxes, having the quality that the components of the syntactic representation (e.g., XML elements) reflect the components in the original schema as closely as possible. For programming APIs, having the quality that the methods and objects provided by the API reflect the schema of the objects of the data being accessed as closely as possible.

empty element tag

In XML, a tag that defines an element that has no syntactic content. It acts as both the start- and end-tag for element.

entity instance

An instance of an EXPRESS entity. Entity instances have identity within a given population and reflect one or more entity types as governed by the constraints on the entity as defined in the governing model.

entity type

In EXPRESS, an object that defines a named set of attributes. One entity type may be the supertype of other entity types, forming a type hierarchy.

EXPRESS repository

A repository containing entity instances governed by EXPRESS models.

late bound

For data representation, having the quality that the components of the representation (e.g., XML elements) reflect the abstract model that governs the data being represented, not the schema of the data itself. For EXPRESS-driven data, the late bound representation of an entity instance would have a name like "entity instance", with the schema-specific information represented through some generic mechanism.

model

In EXPRESS, the set of types and constraints resulting from the combination of schemas. A population of entity instances is governed by a single model. When there is exactly one schema in a model, the model and schema are identical.

partial entity instance

A component of a complex entity instance that exhibits the attributes defined by a single entity type.

population

In EXPRESS, a set of entity instances that conform to a model.

repository

In EXPRESS, a set of entity instances managed as a single unit of storage or management. Note that the definition of "repository" is somewhat vague because it is fundamentally an implementation detail and has no direct correspondence to any component of the EXPRESS schema or instantiation model. In essence, a repository is a thing that contains entity instances. The instances contained may be governed by multiple models. A repository may contain multiple distinct populations of entities, at least in the abstract.

schema

In EXPRESS, the definition of a set of data and entity types and the constraints on those types. A schema governs a population of data instances.

style sheet

Generically, any specification for the rendition of a XML document. Style sheets are usually declarative but may be partially or entirely procedural. The rendered form of the document is usually visual (e.g., print or online presentation), but the notion of "style" has been generalized to include non-visual and dynamic renditions of documents. The two key style sheet mechanisms used with XML today are Cascading Style Sheets (CSS) and eXtensible Style Language (XSL).

tag

In XML, a syntactic marker that defines the start or end of an element. There are three types of tag: start-tags, end-tags, and empty-start-tags. Start tags may contain the specification of element attributes.

XML declaration

The marker at the start of an XML document that identifies it unambiguously as an XML document. The XML declaration has the form `<?xml version="1.0"?>`.

Related Documents

Documents mentioned in this report.

Minutes of PDML XML Early Binding Workshop. URL: TBD

XML Metadata Interchange (XMI): Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF). Object Management GroupPublication Date: October 20, 1998. URL: http://www.omg.org/cgi-bin/doc?ad/98-10-05

Meta Object Facility (MOF) Specification. URL: http://www.omg.org/cgi-bin/doc?ad/98-10-05

UML Proposal to the Object Management Group. URL: http://www.omg.org/techprocess/meetings/schedule/techtab.html#uml

IBM XMI Toolkit. URL: http://www.alphaworks.ibm.com/tech/xmitoolkit

ISO/IEC 10744:1997 Hypermedia/Time-based Structuring Language (HyTime). ISO. URL: www.hytime.org

ISO 10303:11 EXPRESS Language. ISO. URL: Not available online.

ISO TC184/SC4/ WG5 N 228: EXPRESS Semantic Meta-model for ISO 10303-11:1994*. Phil Spiby, Eurostep UK, Ltd., phil.spiby@eurostep.com. URL: http://www.nist.gov/sc4/archive/wg5/n228/

Colophon

This document generated from the original SGML using DSSSL and the Jade DSSSL engine (www.jclark.com/jade).